NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

UTILIZATION OF A BUBBLE MEMORY SYSTEM AS A MICROCOMPUTER DISK RESOURCE

by

Gary A. Theis

March 1984

Thesis Advisor: M. L. Cotton

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Utilization of a Bubble Memory System as a Microcomputer Disk Resource		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis March 1984
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gary A. Theis		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE March 1984
		13. NUMBER OF PAGES 107
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Bubble memory, iSBC 254, Disk Resource, CP/M-86

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Bubble memory is an emerging technology that is only beginning to realize it's potential. The unique properties that this memory system possesses provides advantages in many situations. Bubble memory is non-volatile, solid state, and very durable. In addition this memory has a high density and a fast access time. These attributes are excellent for the non-ideal conditions found in industry and the military.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

This thesis presents an implementation of an iSBC 254 Bubble Memory System as a disk resource in a standard microcomputer environment. An Intel 8086 microprocessor is used as the host executing under Digital Research's CP/M-86 operating system. This implementation is completely transparent to the user and requires no additional disk commands.

S N 0102- LF- 014- 6601

Approved for public release; distribution unlimited

Utilization of a Bubble Memory System as a Microcomputer Disk Resource

by

Gary A. Theis
Lieutenant Commander, United States Navy
B.S., University of Mississippi, 1972

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1984

ABSTRACT

Bubble memory is an emerging technology that is only beginning to realize it's potential. The unique properties that this memory system possesses provides advantages in many situations. Bubble memory is non-volatile, solid state, and very durable. In addition this memory has a high density and a fast access time. These attributes are excellent for the non-ideal conditions found in industry and the military.

This thesis presents an implementation of an iSBC 254 Bubble Memory System as a disk resource in a standard microcomputer environment. An Intel 8086 microprocessor is used as the host executing under Digital Research's CP/M-86 operating system. This implementation is completely transparent to the user and requires no additional disk commands.

TABLE OF CONTENTS

DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis will be listed below, following the firm holding the trademark.

Intel Corporation, Santa Clara, California:

Intel	MULTIBUS	INTELLEC MDS
iSBC 201	Intel 8086	ISBC 86/30
	iSBC 254	

Digital Research, Pacific Grove, California:

CP/M-86 CP/M

I. INTRODUCTION

Magnetic bubble memory is a non-volatile, high density, reliable memory system. It is superior in many ways to conventional secondary storage systems. It's resistance to oppressive environmental conditions is a strong impetus for bubble memory's continuing growth. In addition, other characteristics enhance this memory's value to the marketplace. It's advantages make bubble memory a viable alternative in many situations.

The objective of the work presented here is to demonstrate the utility of a bubble memory system in a conventional operating system (CP/M-86) using a commercially available microprocessor (Intel 8086).

The stated objective is accomplished in two phases. First a basic I/O driver is developed to exercise the iSBC 254 Bubble Memory System. This driver tests critical operations necessary for additional development. All functions significant in data transfer operations are tested for proper operation. The basic driver program also provides a medium for software development and debugging. The next step involves the incorporation of the bubble memory into CP/M-86 as a disk resource. This task requires altering the BIOS portion of the operating system. A new BIOS was generated containing the necessary bubble memory subroutines

in a modularized format. The implementation as a disk is entirely transparent to the user. Procedures for utilizing the bubble memory do not differ from a typical disk system.

Additionally, chapter 2 discusses in some detail the theory of bubble magnetic domains. This section also gives background on typical bubble memory system development and status. A thorough description of the hardware utilized in this thesis is given in chapter 3. The developmental system is functionally outlined and the iSBC 254 bubble memory board is described in detail.

II. BUBBLE MEMORY THEORY

A. BUBBLE DOMAIN THEORY

Bubble domains are small, magnetized, mobile regions within sheets or films of certain magnetic materials. This magnetic "bubble" is a physical phenomenon not unique to any one class of chemical compositions. Certain elements and their alloys, notably iron, cobalt and the rare earth elements, exhibit the property of ferromagnetism. Presently, nearly all bubble devices are made with single-crystal films of multicomponent magnetic rare earth-iron oxides having the garnet structure [Reference 1]. Ferromagnetism permits the material's atoms to exhibit a high degree of alignment despite the natural tendency toward random arrangements. The rule of opposites attracting comes into play in bubble memory technology. The domains existing in a substance are magnetized in either a positive (up) direction or negative (down) direction. In the absence of an external field the domains interact with one another, resulting in zero net magnetism. The more "up" domains you have, the more strongly they interact with those pointing "down," causing the bubble to grow larger [Reference 2]. An opposite force to this ballooning effect occurs at the wall of the bubble, where domains are in various stages of pointing down, crossways, and up (see Figure 2.1). This area of transition

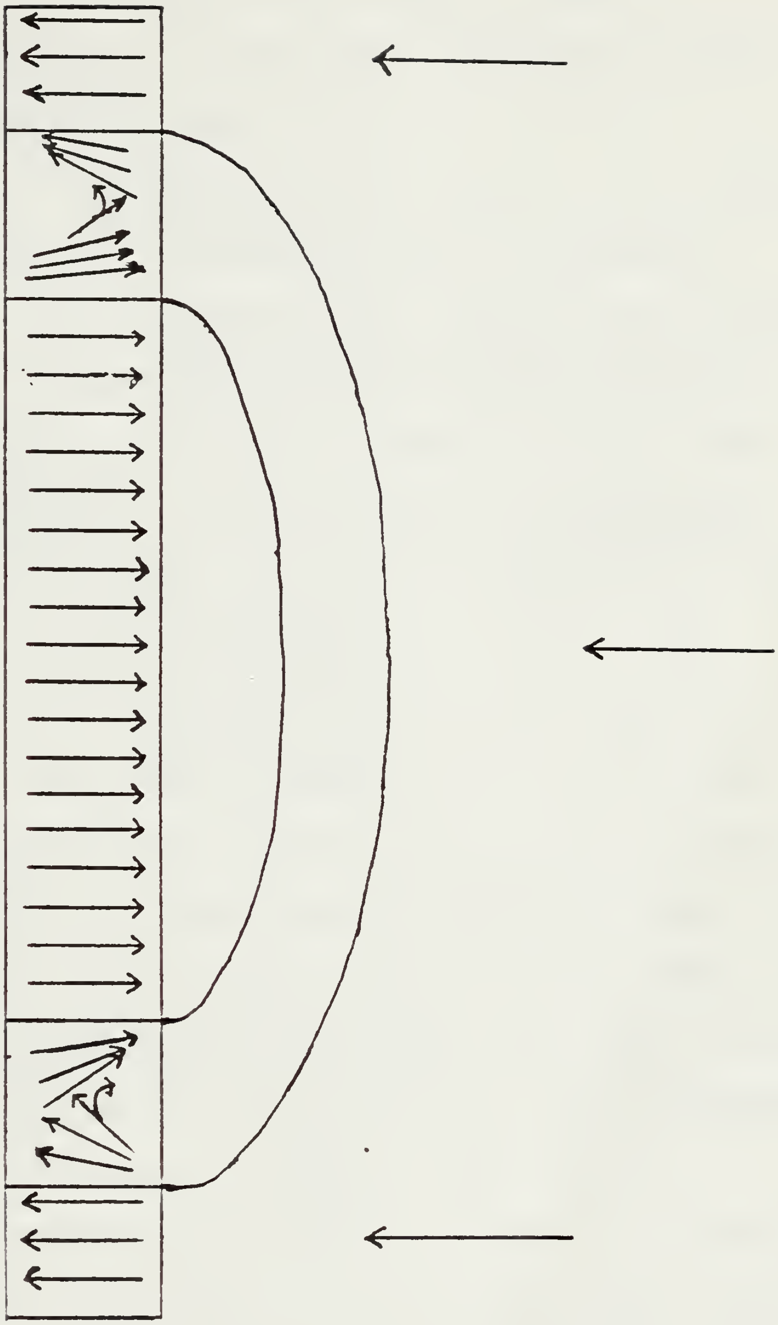


Figure 2.1
BUBBLE DOMAIN MAGNETIZATION

bounding the bubble tends to widen and slow the bubble's growth. The forces never balance and the bubble either continues to grow or collapses upon itself. Thus the ferromagnetic substance is a continually changing pattern of serpentine strips (see Figure 2.2a).

When the magnetic substrate is cut properly into a thin, flat wafer, the domains jut perpendicularly through the plane of the chip. Their positive ends pointing up or down (see Figure 2.2a). Making bubbles stable (and useful) is accomplished by applying an external magnetic field. The strip domains magnetized in the direction of the magnetic field will increase in volume while those magnetized in the opposite direction will shrink (see Figure 2.2b). The domains will continue to be reduced until they completely disappear or until they reach a specific size (see Figure 2.2c). The strength of the external magnetic field is the determining factor [Reference 2]. In actual bubble memory devices, the domains shrink until they are approximately .0001 inch wide. When viewed from above using a microscope they appear round, hence the bubble designation [Reference 3]. This phenomenon is the result of the process of energy minimization.

The applied external field, the bias field, is essential for bubble stability. As long as this field is kept constant, the bubbles neither expand or contract and are held at an acceptable size. The strength of the bias field necessary to maintain stability is of the order of 100-200 Oersteds.

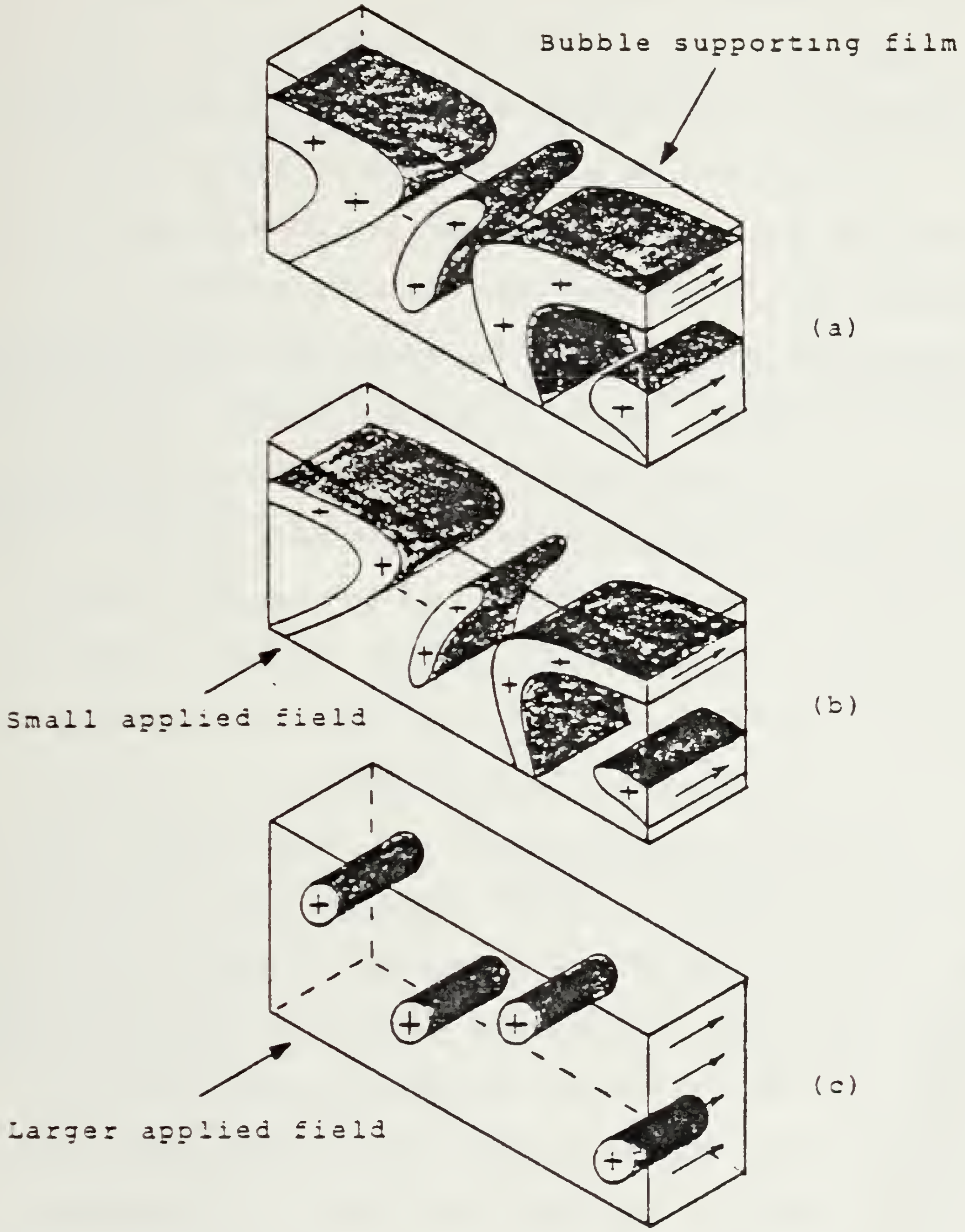


Figure 2.2

(a) SERPENTINE STRIPS, (b) MAGNETIZED STRIPS, (c) CYLINDERS

Small permanent magnets can easily supply the field strength required. These permanent magnets are immune to power fluctuations and are the reason that bubble memory is non-volatile. The stable equilibrium of the bubble domains is the result of a combination of three forces. The domain is preserved by its own magnetization acting against that of the external field. The internal forces produced counteracts the squeezing forces of the bias field. The circular shape is preserved by the magnetic surface tension located at the domain walls [Reference 3].

In order to produce an operational memory system, the bubble domains had to be moved through the substrate in an orderly fashion. Moving bubbles requires setting up a magnetic field gradient within the plane of the chip. This magnetic gradient unbalances the stability of the bubble. The domains will then move through the substrate toward any position that minimizes energy. A permaloy (nickel-iron alloy) track can be bonded to the surface of the substrate (see Figure 2.3). The bubbles will move along these tracks when the magnetic gradient is applied in a specific manner. At a designated point where a detector is located, the presence of a bubble can represent a binary 1. The absence of a bubble represents a binary 0. This magnetic detection is similar to conventional magnetic devices. The distinguishing feature is the fact that no mechanical moving parts are present. This factor allows a bubble memory

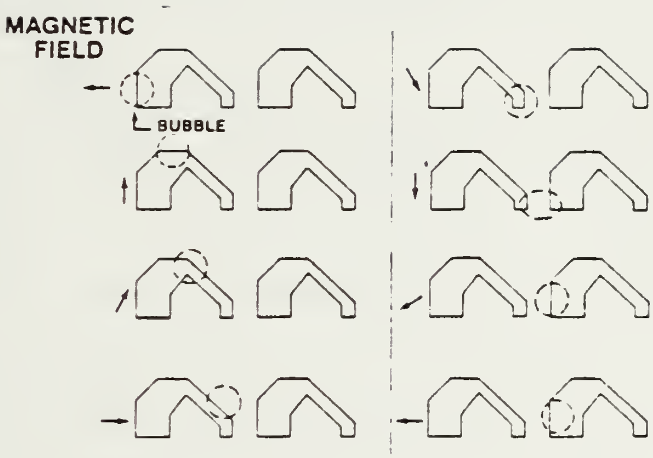


Figure 2.3
BUBBLE PROPAGATION

system to be entirely solid state. Bubble memory's nonvolatility makes it useful for almost any situation in which data that is being stored must be maintained. Nonvolatility also makes bubble memory portable. A user can remove the device from one computer, transport it and find all data intact. The fact that this memory is not electromechanical adds to its reliability and durability. [Reference 2] The next section will describe how bubble theory has been applied in memory engineering.

B. APPLICATION OF BUBBLE THEORY

This section will describe the general designs of bubble memory devices. The basic operations necessary to support a system are, bubble propagation, bubble generation, and bubble detection [Reference 1]. In addition to these basic functions, the data must be organized in such a way as to minimize access time.

1. Bubble Propagation

As previously mentioned, the bubble domains will move in the presence of a magnetic field gradient. A rotating bias field set within the plane of the chip accomplishes this task. The chip is wrapped with two crossed wire coils and the appropriate current is passed through them. By rotating this field, known as the drive field, a magnetic impulse can be generated through the device. The bubble domains travel with this impulse and thus movement is created [Reference 4].

beginning of the input track. The seed is generated by an electric current pulse in a hairpin-shaped loop of conductive material. The pulse is strong enough to reverse the bias field locally and thus allow a bubble domain to be created. Once having been created, the seed bubble remains in existence as long as the external bias field is maintained. The seed circulates under a permalloy patch, driven by the rotating field. This bubble is constrained to a kidney shape by the interaction of the bias and rotating fields with the metal patch (see Figure 2.4). The seed is split in two by a current pulse in the hairpin-shaped conductor. One of them remains under the patch as the seed, and the other is driven by the rotating field onto the input track section of the chip. The current pulse that splits the seed is generated to store a binary 1 in memory. To store a binary 0, the pulse is omitted. This seed bubble process is extremely temperature sensitive. A memory system must be able to vary the current pulse over a range large enough to compensate for large temperature variations.[Reference 4]

3. <u>Bubble Detection</u>

A bubble detector is essentially a magneto-resistive bridge formed by interconnecting the permalloy chevrons to make a continuous electrical path of maximum length. As bubbles pass under the bridge, the resistance changes slightly, modulating the currents through the bridge and creating an output voltage of several millivolts. Bubbles

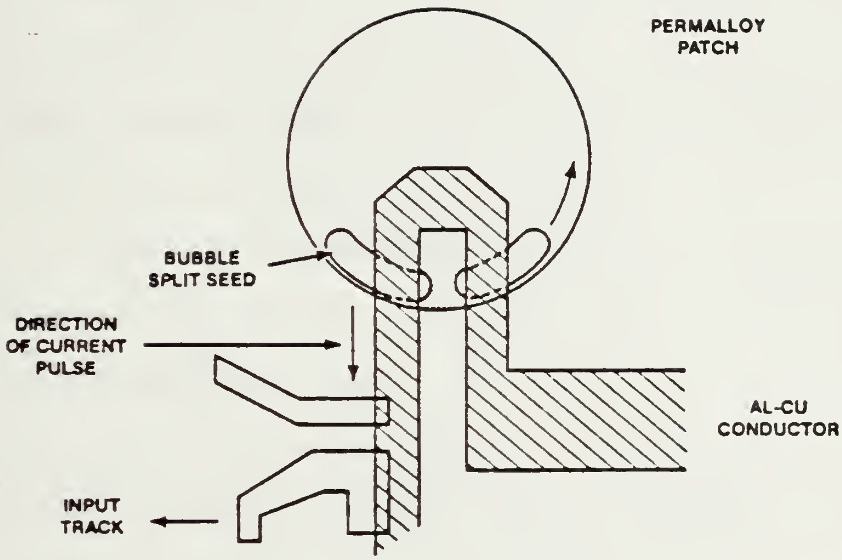


Figure 2.4
BUBBLE GENERATOR

are stretched at right angles to the direction of propagation by adding parallel rows of chevrons. These stretched bubbles generate larger output signals at the detector. Beyond the detector the bubbles run into the guard rail and they are annihilated (see Figure 2.5) [Reference 4].

4. Bubble Architecture

The architectures presented here follow one another in the historical development of bubble memory systems. Each improved the data transfer rates critical to acceptance as a viable memory source.

a. Shift Register Configuration

The shift register architecture suffers from two fundamental problems: (1) If a single defect exists in the shift register chain, the entire chip is bad; and, (2) Data must be cycled through the entire chip in order for the user to gain access to what is needed. If the device is a 1M bit chip and the information is stored halfway down the chain, all the bubbles must move half the length of the chain, in this case 500,000 steps. A typical circulating frequency is 200kHz. In the above example it would take over 2 seconds to access the data desired [Reference 2]. This clearly is unacceptable in modern computer systems (see Figure 2.6a).

b. Major-Minor Loop Design

The problems of the shift register approach were alleviated by employing major and minor loop

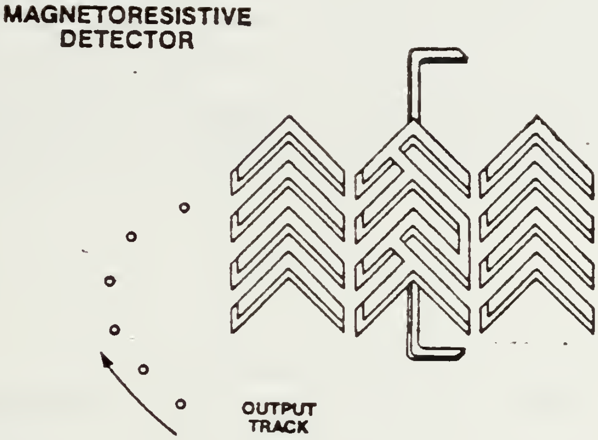
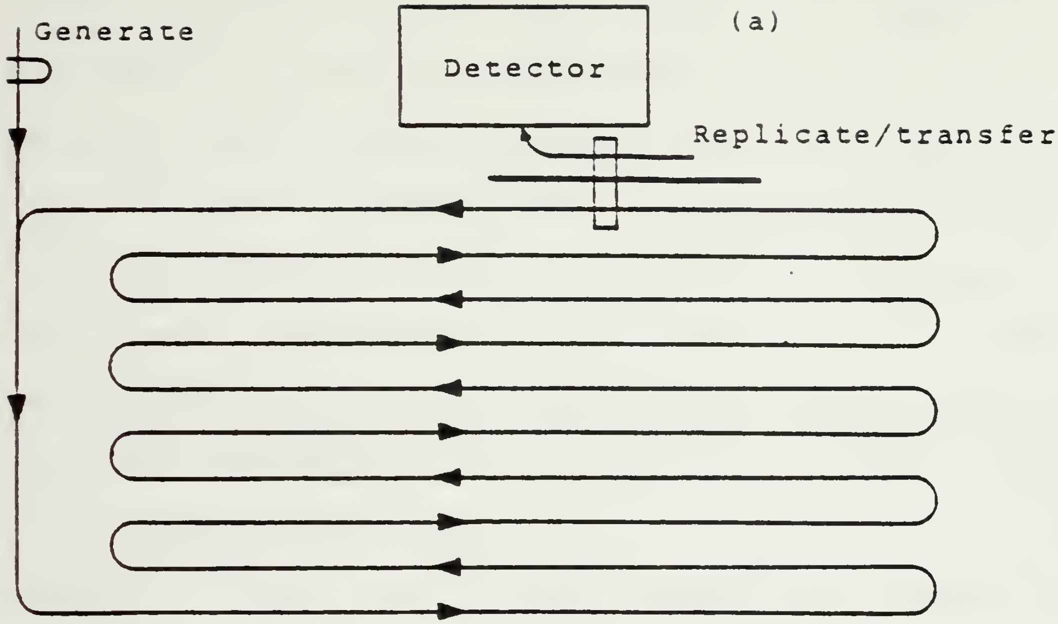


Figure 2.5
BUBBLE DETECTOR



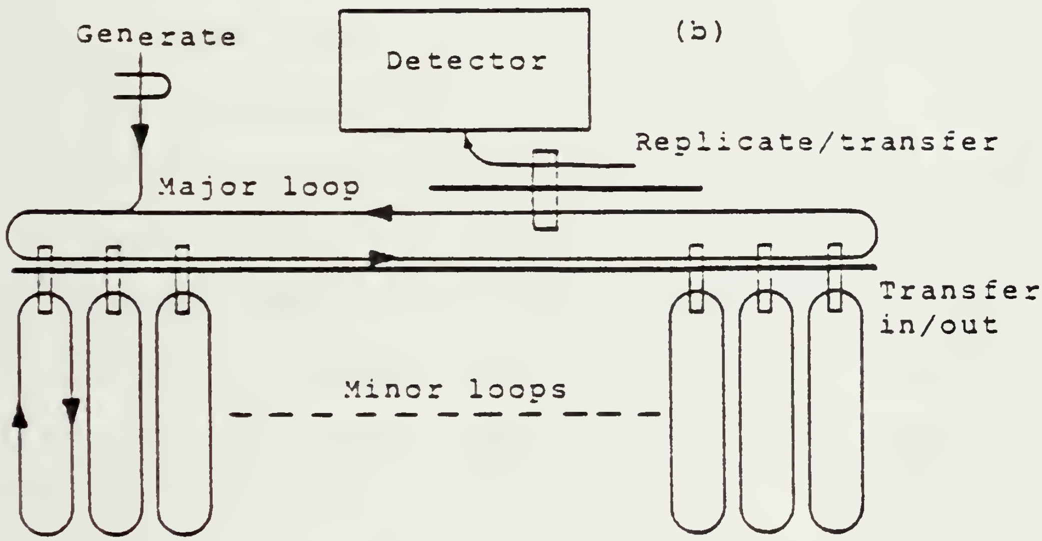


Figure 2.6

(a) SHIFT_REGISTER ARCHITECTURE,
(b) MAJOR/MINOR-LOOP ARCHITECTURE

architectures (see Figure 2.6b). In this configuration, data is stored in minor loops. When a read function is initiated, the data is rotated onto the major loop, detected, and recycled back onto the minor loop position where it began. Access times were improved greatly using this architecture, but improvement was still needed. An additional advantage of this configuration was in the area of chip production. Ths manufacturers were now able to provide redundancy in the number of minor loops. The extra loops that were added provided a margin of error in chip defects. If one loop was bad, an extra loop could take its place. Typically manufacturers provide up to 25 additional loops to compensate for any defects.

c. Block Replicate Architecture

The major-minor loop design improved accessibility but problems still existed. The fact that the bubble domains had to be recycled to their positions retarded access time. The block replicate architecture solved this problem (see Figure 2.7). This configuration involves swapping and replicating the bubble domains. When data is written into the system, bubbles on the input track are swapped with old data on the minor loops. The old data is then destroyed. When reading the minor loop, data is replicated onto the output track. The data remains intact in the minor loops. Swapping occurs when a current pulse, in a conductor under the chevrons, causes the bubble to

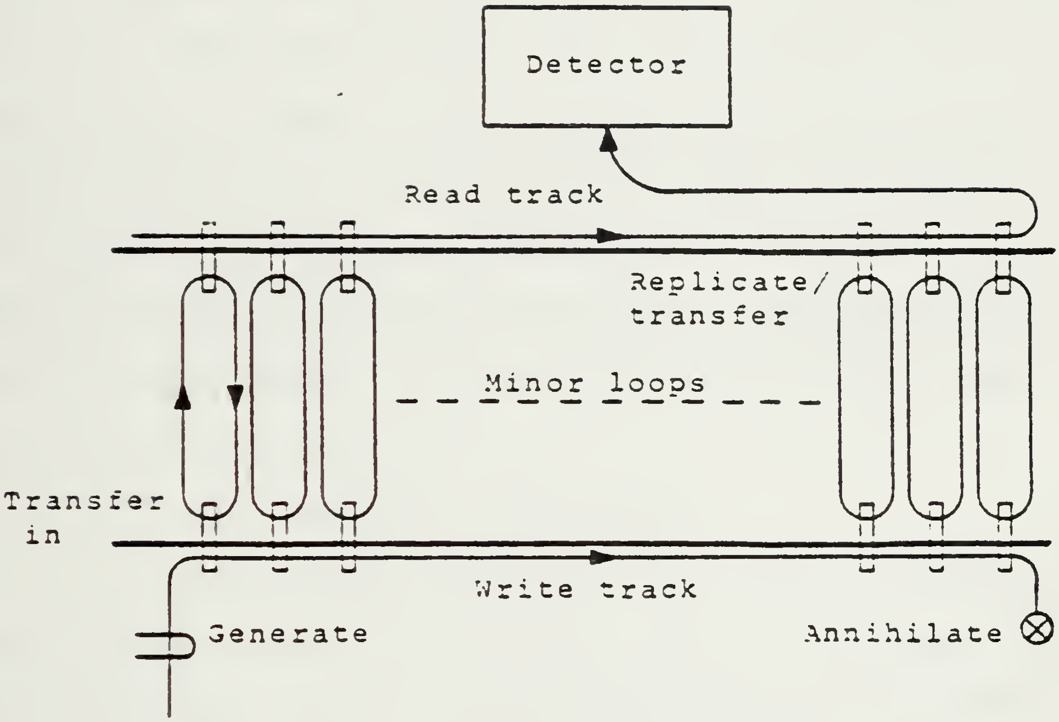


Figure 2.7
BLOCK/REPLICATE ARCHITECTURE

jump from the input track to the storage loop and vice versa. The swap pulse is essentially rectangular, preserving the bubble without cutting it in two. For replication, the bubble is propagated under a large element where it is stretched out. As it passes under a hairpin shaped conductor loop it is cut by a current pulse just as in bubble generation. The replicating current pulse waveshape has a high, narrow leading spike for cutting the original bubble in two, and a lower, wider trailing portion during which the new bubble moves under the output track. This pulse lasts one-quarter of a cycle. In this manner data is propagated to be read, yet retained in the minor loops for storage [Reference 4].

d. Odd-Even Loop Architecture

A variation upon the block replicate design improves access time even further. Due to the properties involving bubble domain interaction, a domain can exist only in alternate positions. This space in between each data position means that data can be manipulated only every second cycle. A way around this problem was found in the odd-even loop architecture (see Figure 2.8). The minor loops are divided into even and odd sections. On one cycle the even bits are read and on the next cycle the odd bits are read. The positions of these bits are staggered and they are interleaved on the way to the detector. The write operation is similarly performed except no interleaving is needed [Reference 6].

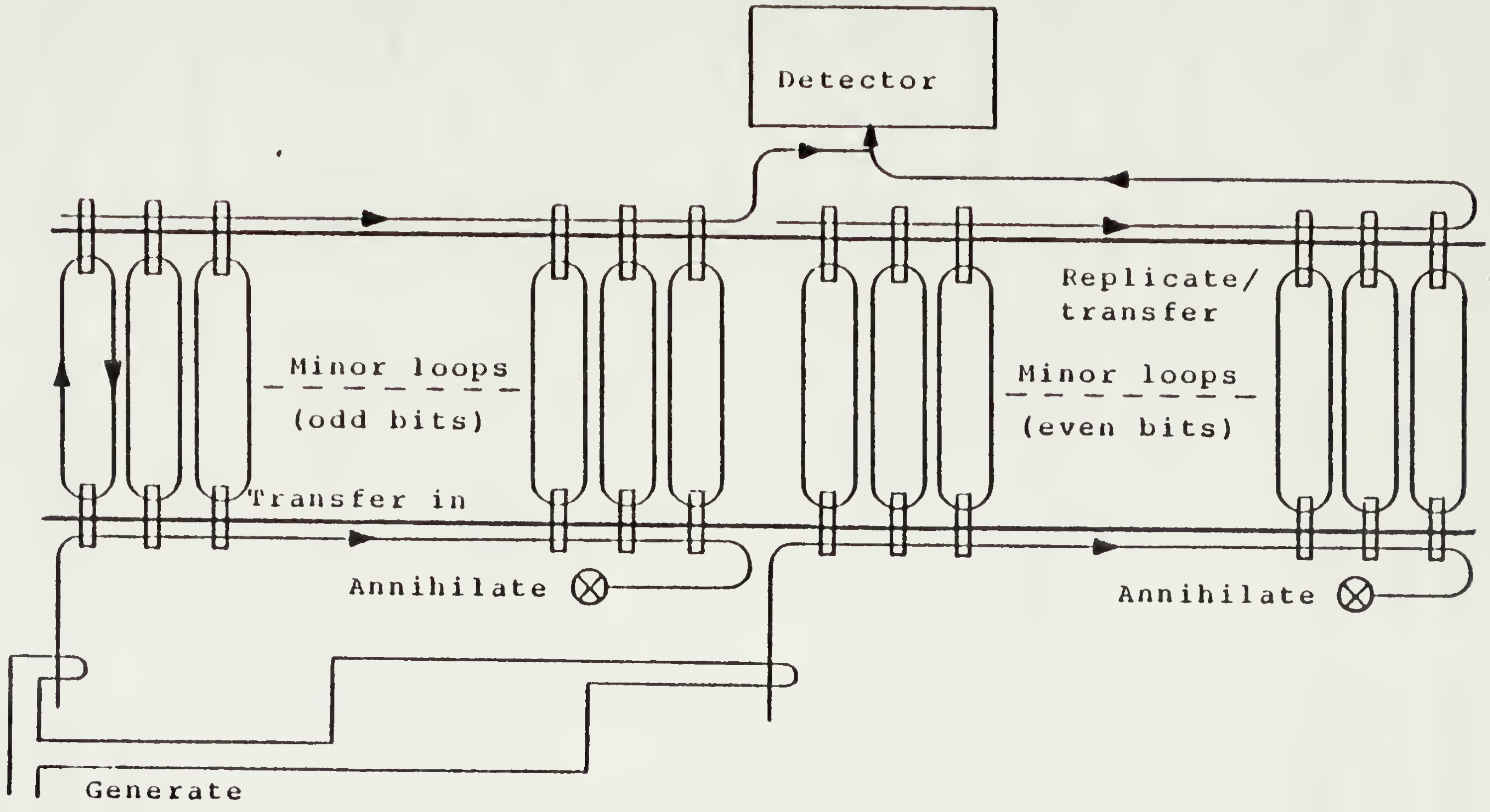


Figure 2.8
BLOCK/REPLICATE ODD/EVEN ARCHITECTURE

As a result of these architectural improvements, access time has been cut from 2 seconds (shift register) to 40 milliseconds. While this time is 1,000 times slower than ram memory, it is 2 to 4 times faster than either hard or floppy disk.

C. PRESENT BUBBLE MEMORY STATUS

The market for bubble memory has never materialized as anticipated when it was first introduced. Essentially, bubble memory has been playing catch up for the past 15 years. In the late 1960's bubble technology was seen as the answer to unwieldy core memories and slow disk systems then in use. Research continued at Bell Labs as well as at IBM, National Semiconductor, Motorola, and Texas Instruments. Mastering bubble technology was no easy task however. As the companies struggled to get their product out of the labs and into production, they neglected to develop supporting electronics. This circuitry, notably bubble memory controllers, is essential to make bubble memories as easy to use as disks. This lapse alone cost the industry two or three years in terms of market acceptance. In addition the price of semiconductor memories and disk systems continued to fall. As a result, bubble memory sales plummeted [Reference 7]. This lack of sales volume resulted in the price of a bubble system remaining very high in comparison to its competitors. One by one

companies dropped out of the market, until Intel Corporation was the lone producer in the United States. At one point there was estimated to be 200 engineers working on bubble memory compared with 50,000 individuals researching silicon memories. The lull was broken in 1979 with the advent of Intel's 1 megabit device. The initial price was a stiff $2,500. Production costs have since been reduced sufficiently to allow a $300 current pricetag. Although it is doubtful if bubble memory will ever displace disk systems, it has found a growing segment in today's marketplace. Its solid state durability has made it a natural selection for systems in harsh environments. Bubble systems have also found their way into a few personal computer systems. With the coming of Intel's 512k byte chip later this year and a 2-megabyte in 1986, the market should open even further. Today bubble memory seems to have come back from near disaster. It is now viewed with enthusiasm as a young technology with an as yet unknown potential. The following chapters will describe how a particular bubble memory system, the Intel iSBC 254, can operate in a microprocessor environment.

III. HARDWARE SYSTEM DESCRIPTION

A. OVERVIEW

The major components used in the work described here consist of an iSBC 254 bubble memory system, an iSBC 86/30B single board computer, an Intellec single density MDS, and an iSBC 201 single density disk controller. The system was operated using the CP/M-86 operating system (version 1.0 as modified in Reference 13). The following sections will describe each component. Particular emphasis will be placed on the bubble memory system.

B. iSBC 254 BUBBLE MEMORY BOARD

The iSBC 254 bubble memory board is a fully assembled, multibus compatible, non-volatile memory. The board is capable of utilizing up to four Intel 7110 bubble memory modules. The rotating field operates at a frequency of 50Khz. A permanent magnet provides the bias field of 20 oersteds. The operating temperature range is between 0 and 50 degrees centigrade with 100 FPM of airflow. The iSBC 254 is compatible with 16-bit addressing for 8 bit microprocessors and 20 bit addressing for 16 bit machines. There are three modes of data transfer available: polled, interrupt, and DMA.

The iSBC 254 configuration used in this work consisted of two 7110 modules, their support components, one controller, a DMA controller, and associated Multibus interface I/O circuitry. This configuration has a maximum data transfer rate of 25K bytes/sec with an average access time of 48ms. A storage capacity of 256K bytes of non-volatile read/write memory is available. The 7220-1 BMC controller interfaces the memory modules to the multibus circuitry via I/O buffers. These buffers then transfer data, address, control, and status information to the system bus and iSBC 254 board. No special timing considerations or hardware modifications were necessary. The iSBC is fully compatible with any Intel host computer on a Multibus system. No attempt will be made here to explain the complex internal timing and operations. A full explanation can be found in Reference 8. Instead, a brief outline will be given on the operation of each of the major board components.

The following devices will be described as to function and system interface:

1. 7110 Bubble Memory Module
2. 7220-1 Bubble Memory Controller (BMC)
3. 7242 Formatter/Sense Amplifier (FSA)
4. 7230 Current Pulse Generator (CPG)
5. 7250 Coil Predriver (CPD)
6. 7254 Quad VMOS Drive Transistor
7. Power Fail Circuitry

1. Component Functions

The 7110 magnetic bubble module is a high density, 1 megabit solid state memory chip. The MBM holds the bubble data for storage and transfer. The architecture is odd-even, block replicate. The 1 megabit storage capacity is provided by 256 loops of 4,096 bits each. When error correcting is selected 14 additional loops are incorporated for the fire code. If error correcting is not implemented 272 loops are used for data. The module itself is divided into four quads. Each half module consists of an "odd" and "even" quad. Odd and even refers to the bit position of the stored data. A half module consists of 160 loops. Since only 135 are required for data and the ECC code, 25 are left for redundancy. In practice the module is screened for up to 24 bad loops to allow the user 16 extra bits if error correcting is not implemented. Each quad has an 81st loop called the bootloop. This loop provides a map indicating the good and bad storage loops. The bootloop is written during production and normally requires no modification. The bootloop also provides synchronization data used as a reference for a physical page address. The data flow, as previously described in chapter 2, is typical of the odd-even block replicate architecture [Reference 8].

The 7220-1 Bubble Memory Controller provides all the timing and control functions needed to operate the system. It is the single point of contact between the host and

memory. The 7220-1 provides a suitable microprocessor interface as well as an interface to the support chips on the iSBC 254 board [Reference 9]. The method of communication with the controller will be discussed later in this chapter.

The 7242 Formatter/Sense Amplifier accepts signals from the bubble detectors in the MBM. During read operations, this device buffers the signals and performs formatting operations. During write operations, the 7242 enables the current pulses of the 7230 that causes the bubbles to be generated. Automatic error detection and correction of the data can be performed by the 7242. The bootloop is automatically placed in the 7242 bootloop register to serve as a data map for the system [Reference 9].

The 7230 Current Pulse Generator supplies the pulses that produce the magnetic bubbles and transfer them into and out of the storage loops of the MBM [Reference 9].

The 7250 and two 7254's supply the drive currents for the in-place rotating magnetic field (X and Y coils) that move the magnetic bubbles within the MBM [Reference 9].

The bubble memory is accessed by passing currents of the proper magnitude and phase through two coils within the MBM. These currents must always be of the proper amplitude and phase or data can be lost. It is also critical to avoid any transient pulses that may occur. The purpose of the power-fail circuitry is to prevent

these transients and to monitor the system voltages. Should power fail, the coil currents must stop in the proper phase [Reference 9].

To better illustrate the interactions between the various components, the data flow within the system will now be explained (see Figure 3.1). During the read operation, bubbles from the storage loops are replicated onto an output track and then moved to a detector within the MBM. All movements and current pulses are under the control of the 7220-1 controller. The magnetic field rotation and timing are also controlled by the 7220-1. The bubble detector outputs a differential voltage according to whether a bubble is present or absent in the detector. This voltage is fed to the detector input of the Formatter/Sense Amplifier. The data path between the 7110 MBM and the FSA consists of two channels connected to the two halves of the MBM. When data is written, the bit stream is divided with half of the data going to each side of the MBM. During a read operation, data from each half of the MBM goes to the corresponding channel of the FSA. In the FSA, the sense amplifier performs a sample-and-hold function on the detector input data. The sense amplifier then produces a digital one or zero. The resulting data bit is then paired with the corresponding bit in the FSA bootloop register. If an incoming data bit is found to be from a good loop, it is stored in the FSA

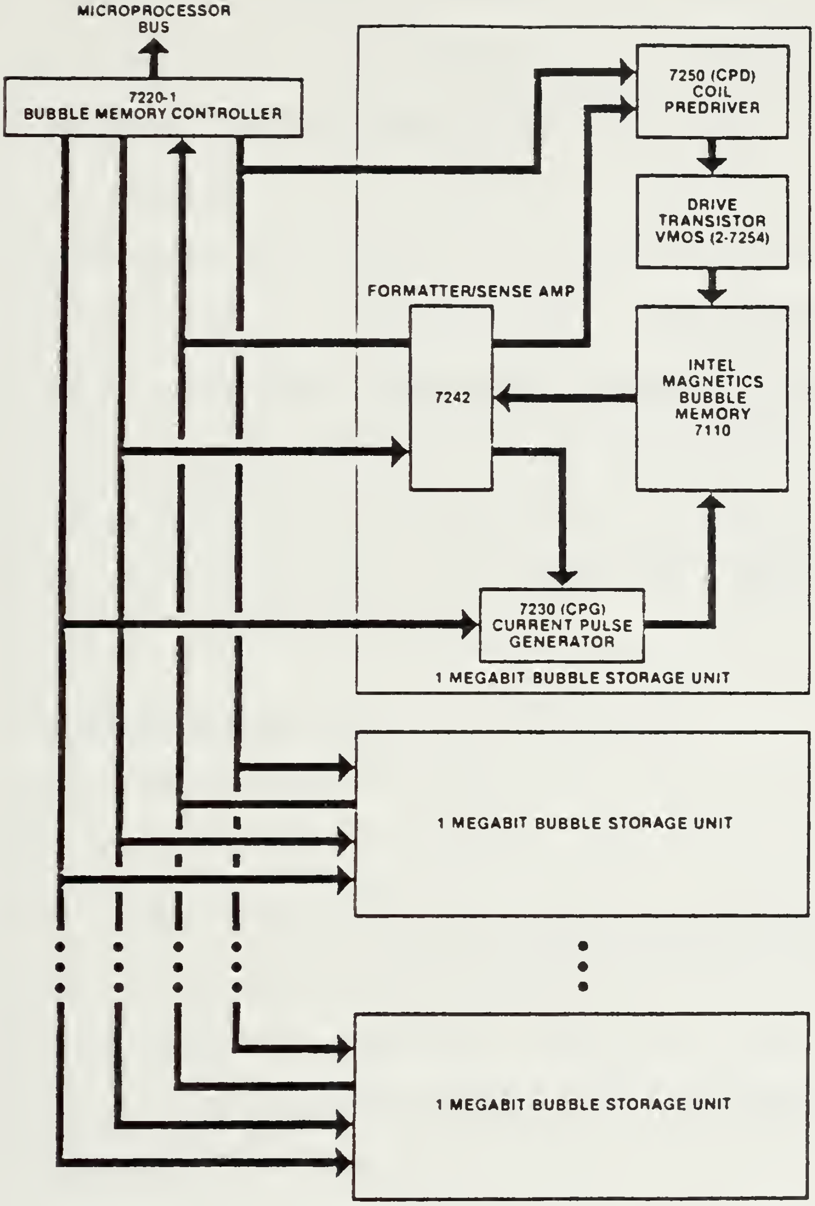


Figure 3.1
SYSTEM BLOCK DIAGRAM

FIFO buffer. Otherwise the data is ignored. This process continues until both channel's FIFOs are filled with 256 bits. Error detection and correction, if enabled by the user, is applied to each block of 256 bits at this point. If error correction is not enabled, 272 bits are used as data. As the data leaves the FSA, the bit patterns are interleaved and sent to the 7220 BMC. The transfer is in the form of a serial bit stream via a one line bidirectional data bus. In the 7220 BMC, the data undergoes a serial-to-parallel conversion and is assembled into bytes that are buffered in the 7220 FIFO. It is from this FIFO that the data is written onto the user interface [Reference 4].

2. Communicating With the 7220-1 BMC

The bubble memory controller is the single point of contact with the host interface. The CPU views the BMC as two input/output ports on the bus. When the least significant bit of the address line is active (A0=1), the command/status port is selected. When the least significant bit of the address line is inactive, the data port is selected. For simplicity the BMC can be viewed as a 40 byte FIFO buffer and 6 eight bit registers. The primary purpose of the FIFO is to reconcile differences in timing between the user interface and FSA interface. The six 8 bit registers internal to the BMC are loaded by the user with information regarding the operation of the system. Loading these registers before any commands are sent is similar to

passing parameters to a subroutine prior to execution. Hence the registers are referred to as parametric registers. Data transferred between the 7220 and the CPU takes place over an 8 bit data bus. The choice as to whether the data is destined for the FIFO or the parametric registers is made through the command/status port. In one case, the actual commands that cause some operation to take place (read, write, etc.), are signified by a command byte with bit 4 set to 1 and the low order nibble containing one of 16 command codes. If bit 4 is zero, the low order nibble is taken to signify a parametric register pointer. For user convenience the 7220-1 contains a register address counter (RAC). The RAC is self incrementing with each subsequent byte of data transferred on the data port. This feature allows the user, after addressing the first parametric register, to load the register values sequentially without addressing each one. After the last register has been loaded the RAC points to the 7220-1 FIFO for subsequent data transfers. The parametric registers are listed in Figure 3.2.

All commands given to the BMC are issued through the command status port to the command register. The sixteen commands available to the bubble memory are listed along with the hex code.

Register Name	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Read/ Write
Utility Register	0	0	0	0	1	0	1	0	R/W
Block Length Register (LSB)	0	0	0	0	1	0	1	1	W
Block Length Register (MSB)	0	0	0	0	1	1	0	0	W
Enable Register	0	0	0	0	1	1	0	1	W
Address Register (LSB)	0	0	0	0	1	1	1	0	R/W
Address Register (MSB)	0**	0	0	0	1	1	1	1	R/W

Figure 3.2
PARAMETRIC REGISTERS

Write Bootloop Register Masked	01h
Initialize	11h
Read Bubble Data	12h
Write Bubble Data	13h
Read Seek	14h
Read Bootloop Register	15h
Write Bootloop Register	16h
Write Bootloop	17h
Read FSA Status	18h
Abort	19h
Write Seek	1Ah
Read Bootloop	1Bh
Read Corrected Data	1Ch
Reset FIFO	1Dh
MBM Purge	1Eh
Software Reset	1Fh

Read, write, abort, and initialize are described in chapter 4. The remainder of the commands are seldom used in normal operation. They are described in detail in Reference 8.

Addressing flexibility is one of the features of the iSBC 254 bubble memory. In the work described here, two modules were available for use. Using the addressing combinations available, the data could be organized into 2,048 pages of 128 bytes each or 4,096 pages of 64 bytes each. The configuration is determined at run time using the block length register and address register. Figure 3.3 lists the various combinations available for up to four modules.

The Block Length Register (BLR) is a 16 bit value divided into two fields: The "terminal count" field and the "channel" field (nfc) (see Figure 3.4a). The terminal count field ranges over the eleven least significant bits and defines the total number of pages requested for a read

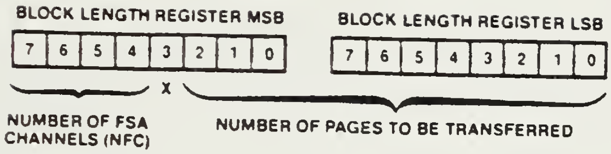
MBM Select AP. MSB Bits (6, 5, 4, 3)	"Channel Field" (BLR MSB Bits 7, 6, 5, 4)				
	0000	0001	0010	0100	1000
0 0 0 0	0	0, 1	0, 1, 2, 3 *	0 to 7	0 to F
0 0 0 1	1	2, 3	4, 5, 6, 7	8 to F	
0 0 1 0	2	4, 5	8, 9, A, B		
0 0 1 1	3	6, 7	C, D, E, F		
0 1 0 0	4	8, 9			
0 1 0 1	5	A, B			
0 1 1 0	6	C, D			
0 1 1 1	7	E, F			
1 0 0 0	8				
1 0 0 1	9				
1 0 1 0	A				
1 0 1 1	B				
1 1 0 0	C				
1 1 0 1	D				
1 1 1 0	E				
1 1 1 1	F				

Figure 3.3

MODULE COMBINATIONS FOR FOUR MODULES

*2 Modules in Parallel (128 Bytes/Page, 2,048 Pages)

(a)



(b)

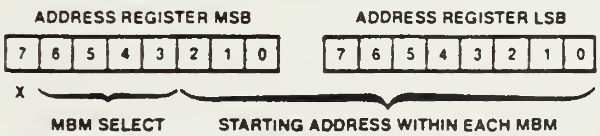


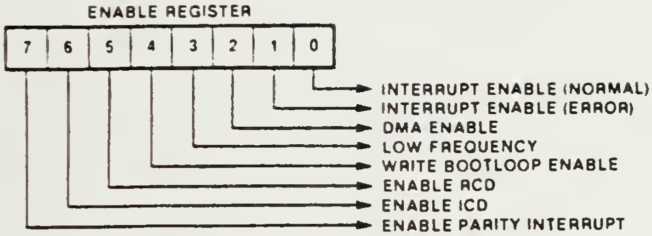
Figure 3.4

(a) BLOCK LENGTH REGISTER BIT FIELD,
(b) ADDRESS REGISTER BIT FIELD

or write operation. With the eleven bits it is possible to request from 1 to 2,048 pages. A field of all zeros indicates a 2,048 page transfer. The channel field indicates the width of the page by specifying the number of channels to be used. A page width of 64, 128, 256, or 512 bytes can be selected (see Figure 3.3). The address register is another 16 bit value containing two fields (see Figure 3.4b). The 11 bit starting address field specifies the page address at which the data transfer begins. If more than one page is transferred the address field is automatically incremented. The second field in the address register, the MBM select field, consists of bits 11, 12, 13, and 14 (bit 15 is not used). These four bits select the particular MBMs to be used in a data transfer. In conjunction with the channel field of the block length register, the MBM select field controls the serial selection of bubble modules or groups of modules operated in parallel.

The final parametric register is the enable register. While the address and block length registers define the system configuration, the enable register defines the mode of operation, interrupt conditions, and error correction level (see Figure 3.5a). The system is capable of three modes of transfer, polled, interrupt, and DMA. This work utilized the polled method which will be explained in chapter 4. A thorough description of the other modes is outlined in Reference 8. The error correction feature can

(a)



(b)

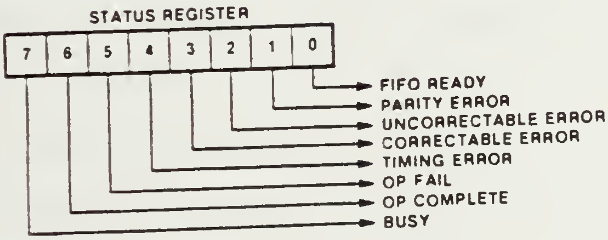


Figure 3.5

STATUS AND ENABLE REGISTERS

be implemented on three levels. Level 1 is the minimum level of error correction. This level is used only when the host is concerned with maintaining bubble integrity. If an error is detected, a read corrected data command is automatically given to the FSA. If the error is correctable, the data transfer continues normally. If the error is not correctable or a timing error exists, the data transfer will be terminated at the error page address. This level is well suited to a go/no-go type of data transfer, and was chosen for the work in this paper. Level 2 is identical to 1 with the exception that upon an uncorrectable error no erroneous data is transferred to the BMC FIFO. Level 3 is the most intensive means of error handling. Under this setting the data transfer is halted if any error is detected. It is by far the most demanding in terms of software requirements [Reference 9].

The final register to be discussed contains the status of the data transfer. Figure 3.5b illustrates the bit designations in the status register. As will be described in chapter 4, this register is extremely important when using the polled method of data transfer. The status register contains information concerning error conditions, command completion (or termination), and the BMC's readiness to accept new commands [Reference 9].

3. Preparing the iSBC 254 Board For Operation

After the board was visually inspected for flaws, the following jumpers were connected:

E79 - E80
E67 - E68
E46 - E45
E30 - E29
E27 - E28
E63 - E64

These jumpers established the base address of 00h, an acknowledge delay period of four clock cycles, serial bus priority, and 8 bit I/O addressing [Reference 8].

The iSBC 254 requires +5VDC at 2.4A and 12VDC at 0.8A. These power requirements are fully compatible with the available multibus power supplies. No hardware modifications are required.

C. DEVELOPMENT SYSTEM

The hardware used in the development of the software, centered upon the Intellec Microcomputer Development System. The Intellec MDS is a coordinated, complete computer system designed around the Intel 8080 microprocessor. The system modules are contained in an eighteen card chassis which features Intel's Multibus architecture. The 8080 microprocessor was removed along with its associated memory modules. The 86/30B was placed in an odd slot to serve as bus master. No additional memory modules were required as the 86/30 has 128K of onboard memory. The iSBC 201 disk controller serviced a dual single density disk drive [Reference 10].

The iSBC 86/30 is a single board microcomputer based on the 16 bit Intel 8086 microprocessor. Included on the board are 128K of dynamic RAM, three programmable parallel I/O ports, programmable timers, priority interrupt control, serial communications interface, and Multibus interface logic.

The CP/M-86 operating system is a product of Digital Research and is produced for use with the 8086 microprocessor. CP/M-86 provides a wide variety of utility built-in commands and transient programs. In addition the user can produce and execute additional transient programs. CP/M-86 provide useful programs for software development. DDT86 is a dynamic programming debugger that proved invaluable for program error correcting. This operating system also provide facilities for writing and editing programs. ED is a very primitive text editor that proved to be adequate for writing the programs used in this work [Reference 11, 12]. The programming language used was 8086 assembly language due to the basic register, data, and I/O port manipulations that were required to make the hardware operate. All programs were written and assembled using this system. No outside resources were required. A printer was available for hard copy transfers.

IV. BASIC SOFTWARE DRIVER DEVELOPMENT

A. DRIVER ORGANIZATION

The first step in developing a working bubble memory system is the development of a basic I/O driver and operationally testing the program for correctness. This task was accomplished using a menu driven program with modular structure and expansion capabilities. The completed program, BUB.A86, is listed as Appendix A.

At the outset of this project, a decision was made to use the polled method of communication. This method was chosen both for its simplicity and reliability. The polled I/O mode is the most simple to implement since no special or external hardware is required to perform data transfers. In the polled I/O mode, the software must determine when to transfer data to or from the FIFO by continually polling a status bit in the BMC's Status Register. This status bit indicates the presence or absence of data in the FIFO on a byte by byte basis. The polled method places the most demand on the host system's processing time since the software continuously must monitor the Status Register [Reference 9].

The driver program's main menu includes the following features:

1. Abort
2. Send Any Command
3. Read Status Register
4. Format Bubble Memory

The sequential development of these features aided in the understanding of the operation of the memory system. In addition this method of development honed the programming skills necessary for subsequent developments. Each subroutine supported succeeding more complex routines.

B. FUNCTION DESCRIPTION

1. Abort Function

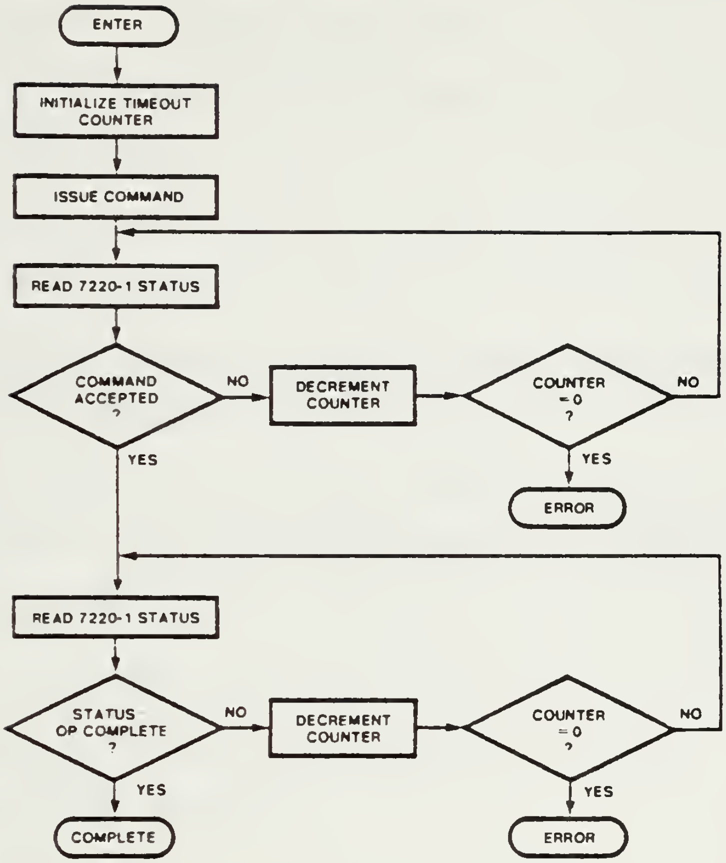
The abort routine was developed initially in accordance with the manufacturers recommendations. When powering the iSBC 254 Bubble Memory Board up for the first time, it is imperative that the board be aborted to insure proper operation. When power is first applied to the iSBC 254, a power fail reset circuit provides a delay (at least 2ms) to allow the 7220-1 bubble memory controller (BMC) to properly power-up. An abort command must then be issued to the BMC in order to reset the system into a known state. After both power supplies have reached 95 percent of their nominal values, a 50ms delay is needed before the abort command is issued to the BMC. This delay could be implemented in software. It was decided however that this interval would be more than taken care of by the amount of

time necessary to power-up the system, load the operating system, and implement the program [Reference 9].

The abort subroutine is a simple example of a non data transfer command sequence (see Figure 4.1). No parametric registers have to be written prior to issuing the command. After the command has been sent, the status register is checked to see if the command has been accepted. This acceptance is signified by the setting of bit 7 in the status register (busy bit). Once the command has been accepted, the status register is checked to see if the operation is completed (bit 6 in the status register) or if it has failed (bit 5 in the status register). If the operation fails or the timeout counter terminates, the routine returns an error message. If the operation is successful, a completion message is sent and the routine returns to the main program. Due to the necessity of aborting the bubble memory, this routine is performed in software automatically each time the bubble memory driver is initiated. It may also be selected for execution from the menu.

2. <u>Send Any Command Function</u>

The second function in the bubble control driver is one in which commands can be sent to the bubble memory controller. Each command developed has its own calling routine which then executes the necessary subroutines. In the work described here it has not been necessary to



COMMENTS: THIS FLOWCHART CAN BE USED TO ISSUE ANY NON-DATA TRANSFER BMC COMMAND BY SUBSTITUTING THE APPROPRIATE COMMAND CODE.

COMMAND	COMMAND CODE
ABORT	19 H
MBM PURGE	1E H
WRITE BOOTLOOP REGISTER	16 H
READ BOOTLOOP REGISTER	15 H
WRITE BOOTLOOP REGISTER MASKED	00 H

Figure 4.1
NON-DATA TRANSFER FLOW CHART

develop all of the bubble commands. Abort, initialize, fifo reset, read, and write were the ones needed to develop the driver. After successfully aborting the bubble memory, it is necessary to initialize it for further operation. This command prepares the bubble system for subsequent operations and is used when the bubble system is powered-up. The parametric registers must be loaded prior to executing this command. The following information is necessary for successful initialization:

The channel field in the block length register must be set to 0001 to arrange all bubbles in the system in a serial configuration. This code allows the individual bootloops to be read from each bubble and then written to the bootloop registers of the corresponding FSA channels.

The MBM select field in the address register must select the last bubble in the system to inform the BMC of the number of bubble modules in the configuration. In the case of the iSBC 254 board that was used in this work there are two bubble modules. Thus the code of 0001 was entered to satisfy this requirement.

The bits in the enable register selecting error corrections must be set in accordance with how subsequent

read and write operations are to be performed. The bubble system must be initialized each time error correction is activated or deactivated. Merely changing error correction levels does not require re-initialization.

The bubble system can be initialized to any address within the module.

When an initialization command is received, all internal registers within the BMC are cleared and the FIFO is reset. The BMC then reads the bootloop from each bubble and writes the corresponding bootloop information into the bootloop registers. The bubble is left positioned according to the value in the address register [Reference 9].

The subroutine used to initialize the bubble system follows the same format as the abort routine with the exception of writing the parametric registers. Since the values to be sent to the registers are constant, they are stored in a table in memory. A routine then moves these values to reserved locations for the parametric register values. Once these parameters are located properly, they are then loaded into the appropriate register. After the initialize command is issued, the polled method then continually reads the status register to insure that the command has been accepted and completed. The applicable messages are printed after the operation.

The read and write command routines were developed after the board was verified to be initializing properly. The read bubble data command causes data to be read from the MBMs and into the BMC's FIFO. Immediately before the read command is issued the parametric registers have to be properly loaded. Since the future plan for the bubble system was to incorporate it into a CPM-86 operating system, the bubble memory was configured to read 128 byte blocks. This arrangement is accomplished by loading the following code into the parametric registers:

The channel field (4 most significant bits in the block length register) contains 0010. This code tells the BMC that both bubble modules are to operate in parallel. Two modules in parallel contain 2,048 128 byte pages.

The block length was set to one for a one page transfer.

The enable register was set for error correction level one.

Bits 6, 5, 4, 3 of the address register's most significant byte were set to 0000. This code addressed the first two modules in the bubble memory system. Although the configuration used here has only two bubble modules, the BMC has the capability of controlling eight.

For this reason it is necessary to specify the correct configuration. The page address can be any value up to 2,047. As a result, this routine can write to any page in the bubble memory system.

The read routine is somewhat more complex than a non-data transfer command (see Figure 4.2). Since the register values can vary, a method was devised to enter the desired numbers from the console. When the read command is selected from the menu, a series of messages will prompt the user to enter the values of the various registers. These values are to be entered in decimal form. A conversion routine changes the numbers to hex code and stores them in the proper memory locations. The registers are then loaded and the read command is issued. Although the routine is set up to read one 128 byte page any appropriate numbers can be entered depending upon how the read command subroutine is tailored. In this case block length is entered as 0001, number of channels is 2, enable register is 32 (20h-level 1 error correcting), the page address can vary from 0000-2047 and the bubble number is entered as 0000. Level one error correcting was chosen to facilitate data transfer in the event of a correctable error. Only a non-correctable error would terminate the operation [Reference 8, 9].

Once the read command is issued, the status register must be checked for command acceptance. When the command is accepted, bit 0 of the status register is viewed

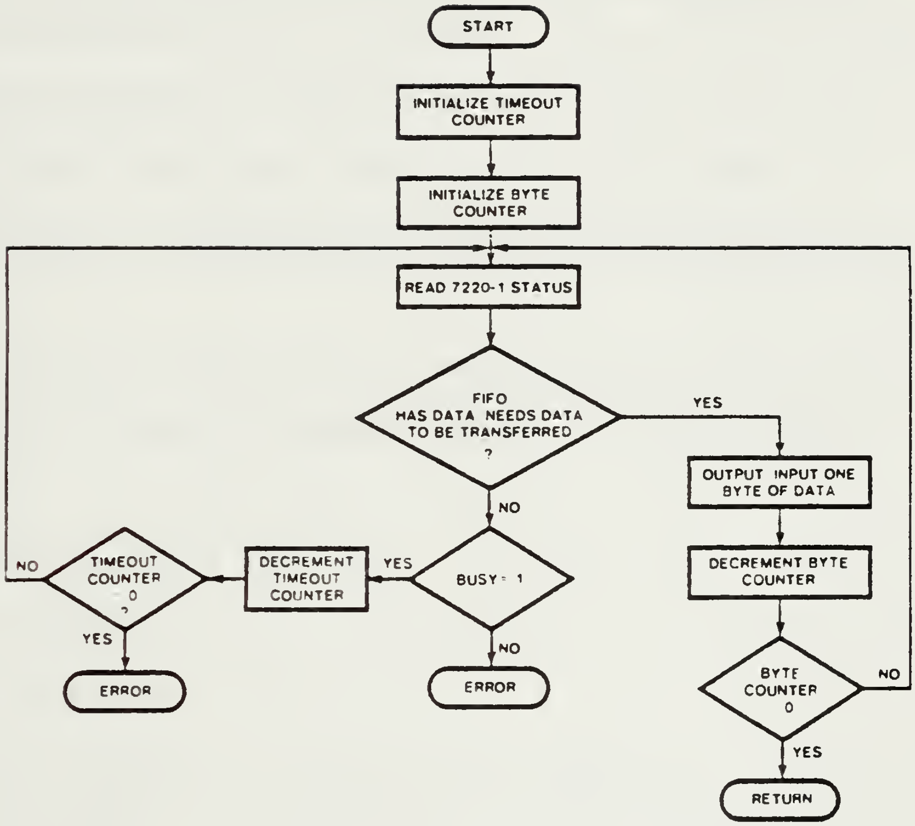


Figure 4.2
POLLED DATA TRANSFER FLOW CHART

to determine if the FIFO is ready to accept data. If the system is ready, data is read one byte at a time. This cycle is repeated for all 128 bytes. If during the transfer the time-out counters run out or an uncorrectable error occurs, the operation will terminate. The status register will signify operation complete (bit 6) when the proper number of bytes have been transferred. This number is determined from the parametric register values described above.

The write bubble data command causes data to be written into the bubble memory modules. A write data transfer does not occur until at least two bytes of data has been written into the FIFO. As in the read routine the parametric registers have to be written with the appropriate data. In this case, the write routine loads 128 bytes into memory. The register parameters are identical to those in the read procedure. The mechanics of the read and write subroutines are virtually identical.

Although most of the commands available to the bubble memory are not developed in this controller, those described above are the most commonly used. Essentially the previously defined commands are all that is needed to produce an effective driver routine [Reference 4]. Provisions have been made in this program for expansion. Any additional command routines can be inserted with a minimum of programming difficulty.

3. Read Status Register Function

The third function in the main program displays the contents of the status register. This feature provides the means of examining the individual bits of the register as they are displayed in binary form. The read and write routine also use this function when returning to the main program. This feature is an excellent diagnostic tool if problems occur in program execution. This routine converts the register bits into ascii 1's and 0's. These characters are then displayed in a message format.

4. Format Function

The last function is used to format the bubble memory to be used with the CPM-86 operating system. Using the existing write routine, format loads each bubble byte with e5h. This code signifies deleted information when operating under CPM. It is essential when the bubble system is used as a "disk" resource.

C. PROBLEMS ENCOUNTERED

The hardware performed in a flawless manner. Everything functioned as expected. The only problems encountered at this juncture originated in software. They were relatively minor, and with the aid of DDT86 the difficulties were quickly resolved. The source of all error could be attributed to programmer inexperience regarding 8086 assembly language programming.

V. INCORPORATION OF THE BUBBLE MEMORY AS A DISK RESOURCE

A. CP/M-86 STRUCTURE

The CP/M-86 structure consists of three parts: The console command processor (CCP), the basic disk operating system (BDOS), and the basic input-output system (BIOS). The CCP interprets commands entered by the user and issues responses. This portion of the system examines command lines typed by the user, performs some simple validation, and calls the appropriate BDOS and BIOS functions. The BDOS contains the various utility routines for managing disks. It makes disk file management transparent to the user. Disk files are often widely scattered in small blocks throughout the storage device. BDOS manages these blocks, dynamically allocating and releasing storage as necessary. The BIOS contains the various drivers that send data to and from the devices, and it receives status information about the success or failure of I/O operations. The CCP and BDOS occupy approximately 10k bytes of memory. These portions are provided on the distribution disk as CPM.H86. The BIOS is modifiable by the user and occupies a variable amount of memory. A skeletal BIOS is provided by Digital Research for operation with disk peripherals [Reference 11]. The next section deals with the modification of this BIOS to accept the iSBC 254 Bubble Memory as a "disk."

B. BIOS MODIFICATION

The CCP and BDOS communicate with physical devices via a well-defined interface in the BIOS. This interface is a set of call and return parameters for the specific functions used. The BIOS modified for bubble usage was based on the one outlined in Reference 13. That BIOS was modified to operate an iSBC 201 disk controller vice an iSBC 204 as provided in the skeletal BIOS. The structure of this modification was deemed adequate for the needs of the work presented here. Since the crux of this work is to operate the Bubble Memory as a "disk" resource, no major modifications in structure were attempted.

The first step in the BIOS modification was to modify the disk parameter table. This table lists the specified device characteristics.

The CP/M operating systems are designed to utilize a table-driven specification for the physical characteristics of each disk device. The modification of this table is essential for the user to add devices to the system. The disk definition table is generated using the following parameters: Logical device number, first and last sector number on each track, optional skew factor, blocksize, disk capacity, the number of directory entries, the number of checked entries, and number of tracks to reserve for the operating system. The utility program GENDEF will generate the disk parameter tables and the necessary scratch pad and

buffer area needed by the operating system for device communication. GENDEF uses a file labeled [filename].def as an input for execution. This input file contains the disk parameters listed above. GENDEF produces a file labeled [filename].lib to be used with an ASM86 include statement in the system BIOS.

The disk definition parameters used in the BIOS of reference 13 were used for the Bubble Memory BIOS. The number of disks was changed to three and the characteristics of the bubble "disk" were added. The iSBC 254 system readily adapted to the CP/M environment. With the exception of the skew factor no parameters needed to be changed from the standard disk parameters. The skew factor was entered as zero due to the fact that there is no latency time applicable to the bubble memory. In a sequential access the bubble pages will not rotate as a disk does. The bubble memory was set as disk number 2. The first and last sectors were 1 and 26 respectively. The block size was defined as 1024 with disk sizes equal to 243K. The bubble has 64 directory entries and checked directory entries. There are two reserved tracks. The object file for the GENDEF command was named SINGLES.DEF. GENDEF then produced the file labeled SINGLES.LIB. A listing of these two files can be found in Appendices B and C.

The next step involved the actual modification of the BIOS. The name chosen for the bubble BIOS was BUBBIOS and

will be referred to as such. A complete listing of BUBBIOS.A86 is in Appendix D. Since the iSBC 254 must be aborted and initialized prior to operation, a suitable place for insertion into the BIOS had to be found. It was decided to place these subroutine calls in the INIT subroutine. They were located in the "not loader bios" portion of this subroutine. This location was chosen to insure that the code and data segments would be properly initialized prior to calling the bubble subroutines. SELDSK was the next subroutine modified. The number of disks needed to be changed to 3 vice 2. The subroutine HOME was modified such that after the track was set to zero, a comparison was made to determine if the bubble had been chosen. If the bubble memory was selected, a jump was inserted to skip the disk device operations and return from the routine. The read and write routines were modified in a similar fashion. Since the iSBC 254 uses completely different routines then a disk device, both READ and WRITE make a comparison and jump to the appropriate bubble memory routines.

After the existing BIOS routines were modified, new routines for the bubble memory had to be added to the existing code. The following bubble subroutines are utilized in BUBBIOS: bubrd (bubble read), bubwrt (bubble write), abort (abort the bubble), wtreg (load the parametric registers), and initb (initialize the bubble). These routines

were all developed in the Bubble Memory Driver delineated in chapter 4. No major modifications were necessary in the structure.

A problem did exist due to the fact that track and sector as supplied by the operating system did not translate into bubble memory page number. As a result a simple algorithm had to be implemented in the subroutine code. It was mentioned earlier that the bubble system would be configured to operate with the two modules in parallel. The result is a data organization of 2,048 pages of 128 bytes. Since CP/M operating system defines a sector as 128 bytes, this translates to a page in bubble memory. CP/M generates the sector and track values for each device access. Since there are 26 sector per track, the page address for the bubble memory can be determined by multiplying the track value by 26 and adding the sector value. This simple algorithm requires 6 lines of code and is inserted prior to calling wtreg in the bubble read and write subroutines. Another relatively minor modification involved returning to the operating system with "1" or "0" values in the al register. If an error occurs in the subroutines, a 1 is to be returned. This action causes an error message to be transmitted by the system. As a result, the bubble routines were modified to return the appropriate values.

C. PROBLEMS ENCOUNTERED AND PERFORMANCE EVALUATION

1. Problems in Implementation

A major difficulty resulted because of confusion in properly setting the si pointer register. During a write operation, the data segment is equated to the dma segment supplied by the operating system. The problem arose when the si pointer was set to the dma address after the data segment value was modified. The values for the dma address and segment are stored in the initial data segment. When the data segment was equated to the dma segment, the label of dma address then pointed to an incorrect value. This error resulted in an incorrect memory location when writing data to the bubble memory. The problem was resolved by setting the si pointer prior to changing the data segment value.

After BUBBIOS was transfering data properly, the data transfer time was observed to be much slower than disk transfers. Since bubble memory is appreciably faster than floppy disks, an examination of the code for inefficiency was conducted. The problem was found to be in the read and write routines. At the beginning of these routines the bubble was initialized. Since CP/M reads and writes a 128 byte sector at a time, this condition resulted in extremely slow multiple page transfers. The overhead for the initialization process was too great for efficient operation. After researching the operating manuals, it was determined that the bubble did not need to be initialized for each

page transferred. The code was then changed for a single initialization in the INIT subroutine as described in the previous section. The bubble performance improved dramatically and will be outlined in the following section.

2. Performance Evaluation

The CP/M-86 utility programs ED, ASM-86, and PIP were used to evaluate the bubble memory performance. ED.CMD is an object-oriented editor for files. The ED program and target files of 17K and 25K bytes were loaded to both an iSBC 254 "disk" and iSBC 201 disk. Using the resident ED program on each device, the target files were written and read. The results are summarized below.

File Size (Bytes)	iSBC-254 (Sec)		iSBC 201 (Sec)	
	Read	Write	Read	Write
17K	4.8	4.8	12.3	15.9
25K	6.7	4.9	15.2	19.2

Three files were used in the tests with the PIP command. The target files were of 6K, 17K, and 60K bytes in size. The target files and the PIP.CMD file were resident on each device. Each file transfer used the resident PIP program. The results are summarized below.

File Size (Bytes)	iSBC 254 (Sec)	iSBC 201 (Sec)
6K	8.3	17.6
17K	21.5	38.1
60K	44.9	62.5

The final test utilized the ASM86 utility program. A 17K byte file was assembled using same-device resident copies of ASM86, the target file and all of the ASM86 output files. The results follow:

File Size (Bytes)	iSBC 254 (Sec)	iSBC 201 (Sec)
17k	63.6	143

From the test results it can be seen that the bubble memory offers a significant advantage in data transfer rates. Overall the bubble was approximately 50 percent faster than the floppy disk. The more I/O intensive the program is, the greater the iSBC 254 performance advantage becomes over the iSBC 201. The major limiting factor in using the bubble system was in the area of transportability. Although you can remove the iSBC board, all power must be shut down. The floppy disk system has an advantage due to the fact that disks can be changed without power interruption. Since the bubble modules cannot be easily interchanged, the memory capacity is limited. The disk system essentially has infinite memory due to the interchangeability.

VI. CONCLUSIONS

A. IMPLEMENTATION SUMMARY

All the objectives set for this thesis were achieved. The iSBC 254 bubble memory system was successfully implemented and evaluated as a system resource. A driver routine was demonstrated and tested using a conventional microprocessor operating system (CP/M-86) and a commercially available microprocessor (Intel 8086). This implementation was accomplished in such a manner that the bubble system appears as a disk resource. This fact allows the user to exercise the bubble system with no special procedural requirements.

The success of this implementation establishes the applicability of the bubble memory in a number of environments. As a disk resource, the iSBC 254 can now be interfaced with other disk systems as a shared resource within the CP/M-86 operating system. The demonstrated interface with a typical host system suggests the compatibility of bubble memory with a wide variety of similar microprocessor systems.

B. FUTURE DEVELOPMENT AND IMPROVEMENT

The iSBC 254 system has intriguing features not employed in this thesis. The system's DMA capability can be investigated in future efforts. This capability requires

additional hardware, but offers an improved data transfer efficiency. If a host system cannot tolerate the software requirements of other modes of transfer, DMA may prove to be the ideal solution.

The interrupt driven data transfer mode requires less processor overhead than the polled method used here. Since the interrupts must be hardwired, some hardware modifications are required. If the interrupt routines are efficient however, the processor can be freed to perform additional tasks.

Another improvement for the future would be the development of Boot Rom and Loader routines using the bubble memory. This additional software would free the host system from any dependency on conventional disk resources.

C. POSSIBLE APPLICATIONS

It is apparent from this implementation that a bubble memory "disk" is superior to conventional floppy disk drives in the area of data transfer rates. Their solid state construction and environmental tolerance add to the bubble system's advantages. This type of memory system can operate in 100 percent humidity, withstand shocks of up to a 200G force, and withstand temperatures in excess of 65 degrees centigrade. The major drawbacks have historically been high price and limited memory capacity. Although these

obstacles still exist, bubble memory system's costs have decreased significantly in the past three years. In addition, higher density chips promise to greatly increase memory capacity. Intel expects to market a 4 Mbit chip later this year, followed by a 2 Mbyte chip in 1986. Due to the entrenchment of disk systems in the marketplace and continuing increases in disk density, it is doubtful that bubble memory will ever displace the floppy or hard disk. There appears to be a growing demand for bubble memory in specialty applications, however. Harsh environments such as those encountered in heavy industry can be easily tolerated. This fact has made robotics a primary source of utilization. Bubble memory's light weight and compactness is invaluable to portable computer systems where space and weight are at a premium. It's rugged dependability allows the bubble to be used where maintenance is infrequent or not possible. The features mentioned above have drawn a serious interest from the military. The harsh conditions encountered in the field preclude many more delicate systems. It is clear that bubble memory will continue to grow in importance with future developments. As computer systems find more applications outside of the ideal environment, the advantages of bubble memory will present a viable alternative.

APPENDIX A
PROGRAM LISTING OF BUB.A86

```
title 'Bubble Memory Controller'
;
;
        cseg                    ;start of code
        org     100h            ;
;
abtcmd  equ     19h             ;abort command
opcomp  equ     40h             ;operation complete mask
cmdsts  equ     0fh             ;command/status port
datreg  equ     0eh             ;data port
blrpt   equ     0bh             ;block 1 reg point
intcmd  equ     11h             ;initialize command
;
cr      equ     0dh             ;
lf      equ     0ah             ;
etx     equ     03h             ;
;
;
start:
        call    abort           ;Power-up command
;
;**********************************************
;*                                            *
;*  This routine displays the function menu   *
;*  and executes the chosen function.         *
;*                                            *
;**********************************************
cmdex:
        mov     dx,offset msg1   ;point to menu message
        call    prtmsg           ;print it
        call    getchar          ;get console input
        and     al,7fh           ;check console parity
        cmp     al,etx           ;compare input to control
        jnz     inst1            ;
        call    system           ;
inst1:  cmp     al,'1'           ;Abort command
        jnz     inst2            ;
        call    abortc           ;
inst2:  cmp     al,'2'           ;Build any command
        jnz     inst3            ;
        call    sencmd           ;
inst3:  cmp     al,'3'           ;Get BMC status
        jnz     inst4            ;
        call    getstat          ;
```

```
inst4:  cmp     al,'4'          ;Format Bubble
        jnz     cmdex           ;
        call    formatb         ;
        jmp     cmdex           ;
;
;*****************************************
;*                                       *
;*  This is the executive routine to write  *
;*  the parametric registers.            *
;*                                       *
;*****************************************
wtreg:
        call    getval          ;get values from console
        call    mvtble          ;mov the values to proper
        call    wtregl          ;load the values into the
        xor     al,al           ;clear al
        ret
;
;*****************************************
;*                                       *
;*     This routine loads the parametric   *
;*     registers in preparation for com-   *
;*     mand execution.                   *
;*                                       *
;*****************************************
wtregl:
        mov     al,blrpt        ;set pointer to BLR(LSB)
        out     cmdsts,al       ;set RAC
        mov     bx,blklen       ;load block length(termina
;
        mov     al,bl           ;this series of instructio
        out     datreg,al       ;combines block length
        mov     al,nfc          ;and the nfc value
        mov     cl,4            ;to form a sixteen bit
        shl     al,cl           ;word to place in
        or      al,bh           ;the block length
        out     datreg,al       ;register
;
        mov     al,enable       ;send enable reg
        out     datreg,al       ;to bmc
;
        mov     bx,pageno       ;load starting page addres
;
        mov     al,bl           ;this series of instructio
        out     datreg,al       ;combines page address.
        mov     al,bblnum       ;and bubble number
        mov     cl,3            ;to form a sixteen bit
        shl     al,cl           ;word to place in
        or      al,bh           ;the address
        out     datreg,al       ;register
```

```
;*****************************************************
;*                                                   *
;*   The routine aborts the bubble memory            *
;*                                                   *
;*****************************************************
abort:
        mov     cx,0ffffh           ;init time out cntr
        mov     bh,opcomp           ;move 40h to bh
        mov     al,abtcmd           ;load abort command
        out     cmdsts,al           ;send abort command
busy:
        in      al,cmdsts           ;read status reg
        and     al,80h              ;mask for busy
        jz      poll                ;if busy jump to poll
        dec     cx                  ;else decrement time out
        xor     ax,ax               ;clear ax reg
        cmp     cx,ax               ;check time out count=0
        jnz     busy                ;time left,try busy again
        jmp     ret1                ;return error
poll:
        in      al,cmdsts           ;read status reg
        test    al,bh               ;wait for status=40h
        jnz     ret2                ;if operation complete ret
        dec     cx                  ;else decrement time out
        xor     ax,ax               ;clear ax reg
        cmp     cx,ax               ;compare timeout to zero
        jnz     poll                ;try again
ret1:
        in      al,cmdsts           ;return with status fail
        mov     dx,offset msg6      ;abort fail msg
        call    prtmsg              ;print the message
        call    system              ;jump to system
ret2:
        mov     dx,27600            ;delay 100ms
loop:
        xor     ax,ax               ;clear ax reg
        dec     dx                  ;decrement count
        cmp     dx,ax               ;compare timeout to zero
        jnz     loop                ;try again
        in      al,cmdsts           ;get status
        ret
;
;*****************************************************
;*                                                   *
;*   This routine initializes the bubble             *
;*                                                   *
;*****************************************************
initz:
        mov     cx,0ffffh           ;set timeout counter
```

```
        mov     bh,opcomp           ;mov 40h into bh
        mov     al,intcmd           ;load init command
        out     cmdsts,al           ;send it
busy1:
        in      al,cmdsts           ;get status
        and     al,80h              ;check for busy
        jz      poll1               ;no-try again
        dec     cx                  ;decrement timeout
        xor     ax,ax               ;clear ax
        cmp     cx,ax               ;compare timeout to zero
        jnz     busy1               ;try again
        jmp     reta1               ;return error
poll1:
        in      al,cmdsts           ;get status
        xor     al,bh               ;check for op comp
        jz      reta2               ;yes-return op complete
        dec     cx                  ;decrement timeout
        xor     ax,ax               ;clear ax
        cmp     cx,ax               ;compare timeout to zero
        jnz     poll1               ;try again
reta1:
        in      al,cmdsts           ;get status
        mov     dx,offset msg4      ;timeout failure
        call    prtmsg              ;
        ret
reta2:
        in      al,cmdsts           ;get status
        mov     dx,offset msg2      ;operation complete
        call    prtmsg              ;
        ret
;
;*********************************************
;*                                           *
;*  This routine converts the console input  *
;*  to hex values and loads these values     *
;*  into the proper memory locations.        *
;*                                           *
;*********************************************
mvtble:
        mov     bx,offset temptbl       ;set pointer
        mov     dl,04               ;# of digits in blklen
        call    convert             ;convert decimal to hex
        mov     blklen,ax           ;mov hex value to mem add
        mov     dl,01               ;# of digits in nfc
        call    convert             ;convert decimal to hex
        mov     nfc,al              ;mov to proper address
        mov     dl,02               ;# of digits in enable
        call    convert             ;convert decimal to hex
        mov     enable,al           ;mov to proper address
        mov     dl,04               ;# of digits in pageno
```

```
        mov     pageno,ax           ;mov to proper addr
        mov     dl,01               ;# of digits in bblnum
        call    convert             ;convert decimal to hex
        mov     bblnum,al           ;move to proper addr
        ret
;
;*********************************************
;*                                           *
;*  This routine calls the operating system  *
;*  to print the message pointed to by DX    *
;*                                           *
;*********************************************
prtmsg:
        mov     cl,09h              ;
        int     224                 ;
        ret
;
;*********************************************
;*                                           *
;*  This routine will jump from this program *
;*  back to the operating system             *
;*                                           *
;*********************************************
system:
        mov     cl,00h              ;
        mov     dl,00h              ;
        int     224                 ;
;
;*********************************************
;*                                           *
;*  This routine uses the operating system   *
;*  to get the character from the console    *
;*                                           *
;*********************************************
getchar:
        mov     cl,01h              ;
        int     224                 ;
        mov     temp1,al            ;
        ret
;
;*********************************************
;*                                           *
;*  This routine calls abort and initializes *
;*  the bubble memory.                       *
;*                                           *
;*********************************************
abortc:
        call    abort               ;Send abort command
        mov     dx,ds               ;move ds location to dx
        mov     es,dx               ;set es equal to ds
        mov     si,offset table1        ;set source pointer
```

```
        mov     di,offset blklen        ;set dest. pointer
        mov     cx,7                    ;set 7 iterations
        cld                             ;clear direection flag
        rep     movs  al,al             ;load bytes from table
        call    wtregl                  ;Write BMC registers
        call    initz                   ;Send initialize command
        xor     al,al                   ;clear al before return
        ret
;
;*********************************************
;*                                           *
;*  This rountine executes the send any com- *
;*  mand function.                           *
;*                                           *
;*********************************************
sencmd:
        call    getcmd                  ;get command and execute
        xor     al,al                   ;clear al before return
        ret
;
;********************************************
;*                                          *
;*  This routine displays the status byte   *
;*  on the console.                         *
;*                                          *
;********************************************
getstat:
        call    stostat                 ;
        mov     dx,offset status            ;print status byte
        call    prtmsg                  ;
        mov     dx,offset msg13             ;print status msg
        call    prtmsg                  ;
        xor     al,al                   ;clear al before return
        ret
;
wblrm:
        xor     al,al                   ;command not implemented
        ret                             ;
;
rdsk:
        xor     al,al                   ;command not implemented
        ret
;
rdblr:
        xor     al,al                   ;command not implemented
        ret
;
wrtblr:
        xor     al,al                   ;command not implemented
        ret
.
```

```
wrtbl:
        xor     al,al               ;command not implemented
        ret
;
rdfsa:
        xor     al,al               ;command not implemented
        ret
;
wrtsk:
        xor     al,al               ;command not implemented
        ret
;
rdbl:
        xor     al,al               ;command not implemented
        ret
;
rdcd:
        xor     al,al               ;command not implemented
        ret
;
resetf:
        xor     al,al               ;command not implemented
        ret
;
purge:
        xor     al,al               ;command not implemented
        ret
;
reset:
        xor     al,al               ;command not implemented
        ret
;
;*******************************************
;*                                         *
;*  This is the calling routine to write a *
;*  128 byte page into bubble memory.      *
;*                                         *
;*******************************************
bmwrt:
        call    abortc  ;abort and initialize the bubble
        call    write   ;write to the bubble memory
        xor     al,al   ;clear al before return
        ret
;
;*******************************************
;*                                         *
;*  This is the calling routine to read a  *
;*  128 byte page from bubble memory.      *
;*                                         *
;*******************************************
bmrd:
```

```
        call    abortc  ;abort and initialize the bubble
        call    wtreg   ;load the parametric registers
        call    read    ;read from the bubble
        xor     al,al   ;clear al before return
        ret
;
;*********************************************
;*                                           *
;*  This routine reads the status register   *
;*  bit by bit and records the bit value in  *
;*  memory as an ascii 1 or Ø .              *
;*                                           *
;*********************************************
stostat:
        in      al,cmdsts
        mov     bx,offset (status+3)    ;set pointer in bx
        mov     cx,8                    ;set number of loops
again:
        shl     al,1                    ;shift msb to left
        jb      skip                    ;jump if carry = 1
        mov byte ptr    [bx],3Øh        ;store an ascii Ø
        jmp     next
skip:
        mov byte ptr    [bx],31h        ;store an ascii 1
next:
        inc     bx                      ;move pointer up one
        loop    again                   ;loop to beginning
        xor     bx,bx                   ;clear bx
        ret
;
;*********************************************
;*                                           *
;*  This routine gets the values of the para-*
;*  metric registers from the console and    *
;*  stores them in memory prior to conversion*
;*  to hex values.                           *
;*                                           *
;*********************************************
getval:
        mov     dx,offset msg7  ;point to msg
        call    prtmsg          ;print it
        mov     dx,offset msg8  ;point to msg
        call    prtmsg          ;print it
        call    conin           ;get input from console
        mov     temptbl,al      ;mov value to temp table
        call    conin           ;get input from console
        mov     temptbl+1,al    ;move to memory location
        call    conin           ;get input from console
        mov     temptbl+2,al    ;move to memory location
        call    conin           ;get input from console
        mov     temptbl+3,al    ;move to memory location
```

```
        mov     dx,offset msg9  ;print next msg
        call    prtmsg          ;
        call    conin           ;get input from console
        mov     temptbl+4,al    ;move to memory location
        mov     dx,offset msg10 ;print next msg
        call    prtmsg          ;
        call    conin           ;get input from console
        mov     temptbl+5,al    ;move to memory location
        call    conin           ;get input from console
        mov     temptbl+6,al    ;move to memory location
        mov     dx,offset msg11 ;print next msg
        call    prtmsg          ;
        call    conin           ;get input value
        mov     temptbl+7,al    ;mov to memory location
        call    conin           ;get input from console
        mov     temptbl+8,al    ;move to memory location
        call    conin           ;get input from console
        mov     temptbl+9,al    ;move to memory location
        call    conin           ;get input from console
        mov     temptbl+10,al   ;move to memory location
        mov     dx,offset msg12 ;print next msg
        call    prtmsg          ;
        call    conin           ;get input from console
        mov     temptbl+11,al   ;move to memory location
        xor     ax,ax           ;clear ax register
        ret
;
;*****************************************
;*                                       *
;*  This routine converts the decimal values *
;*  input from the console to hex values     *
;*                                       *
;*****************************************
convert:
        mov     temp2,0000      ;clear memory location
        cmp     dl,05           ;see if there are five digi
        jz      conv10k         ;yes-start at 10K
        cmp     dl,04           ;see if there are four digi
        jz      conv01k         ;yes-start at 01K
        cmp     dl,03           ;see if there are three dig
        jz      conv100         ;yes-start at 100
        cmp     dl,02           ;see if there are two digit
        jz      conv010         ;yes-start at 10
        cmp     dl,01           ;see if there is one digit
        jz      conv001         ;yes-start at 1
        call    system          ;jump back to system if zer
conv10k:
        mov     al,[bx]         ;load value at bx pointer
        sub     al,30h          ;convert it from ascii
        mov     dx,10000        ;load dx
        mov     ah,00           ;clear ah
```

```
        mul     dx              ;multiply al by 10000
        mov     temp2,ax        ;store it
        inc     bx              ;increment pointer
conv01k:
        mov     al,[bx]         ;load value at bx pointer
        sub     al,30h          ;convert from ascii
        mov     dx,1000         ;load dx
        mov     ah,00           ;clear ah
        mul     dx              ;multiply al by 1000
        mov     dx,ax           ;store result in dx
        mov     ax,temp2        ;get result from previous
        add     ax,dx           ;add the two
        mov     temp2,ax        ;store the total
        inc     bx              ;increment the pointer
conv100:
        mov     al,[bx]         ;load value at bx pointer
        sub     al,30h          ;convert from ascii
        mov     dx,100          ;load dx
        mov     ah,00           ;clear ah
        mul     dx              ;multiply al by 100
        mov     dx,ax           ;store result in dx
        mov     ax,temp2        ;get total from previous
        add     ax,dx           ;add the two
        mov     temp2,ax        ;store the total
        inc     bx              ;increment the pointer
conv010:
        mov     al,[bx]         ;load value at bx pointer
        sub     al,30h          ;convert from ascii
        mov     dx,10           ;load dx
        mov     ah,00           ;clear ah
        mul     dx              ;multiply al by 10
        mov     dx,ax           ;store result in dx
        mov     ax,temp2        ;get total from previous
        add     ax,dx           ;add the two
        mov     temp2,ax        ;store the total
        inc     bx              ;increment the pointer
conv001:
        mov     al,[bx]         ;load value at bx pointer
        sub     al,30h          ;convert from ascii
        mov     ah,00           ;clear ah
        mov     dx,ax           ;store result in dx
        mov     ax,temp2        ;get total from previous
        add     ax,dx           ;add the two
        inc     bx              ;increment the pointer
        ret
```

```
;*********************************************
;*                                           *
;*  This routine reads 128 bytes from the    *
;*            bubble memory                  *
;*                                           *
;*********************************************
read:
        mov     cx,128              ;128 byte count
        mov     bx,cx               ;save count in bx
        mov     di,offset datbuf;set pntr to buffer
        mov     ax,ds               ;set es equal to ds
        mov     es,ax               ;
        mov     al,12h              ;load read command
        out     cmdsts,al           ;send it
        mov     cx,0ffffh           ;load cx with counter
read1:
        in      al,cmdsts           ;get status
        test    al,80h              ;test for busy
        loopz   read1               ;wait for busy
        jcxz    error1              ;timeout error
        mov     cx,bx               ;load # of bytes in cx
read2:
        in      al,cmdsts           ;get status
        test    al,01h              ;test for fifo empty
        jz      read3               ;yes,check for busy
        in      al,datreg           ;no,get data
        stos    al                  ;store it
        loop    read2               ;try again
        jmp     pause               ;wait for good status
read3:
        mov     dx,0ffffh           ;timeout in dx reg
        test    al,80h              ;check for busy
        jz      skip1               ;
        dec     dx                  ;decrement timeout cntr
        cmp     dx,0                ;compare timeout to 0
        jnz     read2               ;still busy-wait
skip1:  sub     bx,cx               ;bytes transferred in bx
        mov     temp3,bx            ;store byte trans count
        jmp     error2              ;op fail error
pause:
        in      al,cmdsts           ;get status
        test    al,80h              ;check for busy
        jz      contin              ;not busy-send status
        mov     cx,0ffffh           ;set up timeout cntr
pollb:
        in      al,cmdsts           ;get status
        test    al,80h              ;check for busy
        loopnz  pollb               ;wait for busy to clear
        jcxz    error1              ;but not too long
contin:
```

```
        mov     dx,offset msg2  ;op complete msg
        call    prtmsg          ;
        ret
error1:
        call    getstat         ;display status byte
        mov     dx,offset msg4  ;timeout failure
        call    prtmsg          ;
        ret
error2:
        call    getstat         ;display status byte
        mov     dx,offset msg3  ;op fail msg
        call    prtmsg          ;
        ret
;
;*********************************************
;*                                           *
;*  This routine writes 128 bytes into the   *
;*             bubble memory                 *
;*                                           *
;*********************************************
write:
        mov     cx,128          ;# of bytes to write
        mov     bx,cx           ;save in bx
        mov     al,1dh          ;fifo reset cmd
        out     cmdsts,al       ;issue it
        push    bx              ;save bx
        call    wtreg           ;write registers
        pop     bx              ;retrieve bx
        mov     si,offset wrtbuf;set pointer
writea:
        mov     al,13h          ;load write cmd
        out     cmdsts,al       ;issue it
        mov     cx,Øffffh       ;timeout cntr
write1:
        in      al,cmdsts       ;get status
        test    al,8Øh          ;check for busy
        loopz   write1          ;wait for busy
        jcxz    error11         ;timeout error
        test    al,Ø1h          ;check for fifo ready
        loopz   write1          ;wait for fifo
        jcxz    error11         ;timeout error
        mov     cx,bx           ;load # of bytes in cx
write2:
        in      al,cmdsts       ;get status
        test    al,Ø1h          ;check for fifo ready
        jz      write3          ;no-wait
        lods    al              ;yes-get data
        out     datreg,al       ;send data to bubble
        loop    write2          ;go again
        jmp     pause1          ;return good status
```

```
write3:
        mov     dx,0ffffh        ;set timeout cntr
        test    al,80h           ;test for busy
        jz      skip11           ;
        dec     dx               ;dec timeout cntr
        cmp     dx,0             ;compare timeout to 0
        jnz     write2           ;try again
skip11: sub     bx,cx            ;# of bytes trans.
        jmp     error22          ;op fail error
pause1:
        in      al,cmdsts        ;get status
        test    al,80h           ;check for busy
        jz      contin1          ;return staus/op comp.
        mov     cx,0ffffh        ;set timeout
pollb1:
        in      al,cmdsts        ;get status
        test    al,80h           ;check for busy
        loopnz  pollb1           ;wait
        jcxz    error11          ;but not too long
contin1:
        call    getstat          ;display status
        mov     dx,offset msg2   ;op complete msg
        call    prtmsg           ;
        ret
error11:
        call    getstat          ;display status
        mov     dx,offset msg4   ;timeout failure
        call    prtmsg           ;
        ret
error22:
        call    getstat          ;display status
        mov     dx,offset msg3   ;op fail msg
        call    prtmsg           ;
        ret
;
;***********************************************
;*                                             *
;*  This routine presents the command menu for *
;*  the send-any-command function in the main  *
;*  menu.  It also calls the appropriate sub-  *
;*  routines for the chosen commands.          *
;*                                             *
;***********************************************
getcmd:
        mov     dx,offset msg14 ;Print menu for commands
        call    prtmsg           ;
        call    getchar          ;Get console input
        and     al,7fh           ;
        cmp     al,etx           ;Compare with control C
        jnz     cmd0             ;
```

```
        call    system          ;If Control C jmp to system
cmd0:   cmp     al,'0'          ;Write bootloop register mask
        jnz     cmd1            ;
        call    wblrm           ;
cmd1:   cmp     al,'1'          ;Initialize command
        jnz     cmd2            ;
        call    abortc          ;
cmd2:   cmp     al,'2'          ;Read bubble data
        jnz     cmd3            ;
        call    bmrd            ;
cmd3:   cmp     al,'3'          ;Write bubble data
        jnz     cmd4            ;
        call    bmwrt           ;
cmd4:   cmp     al,'4'          ;Read seek
        jnz     cmd5            ;
        call    rdsk            ;
cmd5:   cmp     al,'5'          ;Read bootloop register
        jnz     cmd6            ;
        call    rdblr           ;
cmd6:   cmp     al,'6'          ;Write bootloop register
        jnz     cmd7            ;
        call    wrtblr          ;
cmd7:   cmp     al,'7'          ;Write bootloop
        jnz     cmd8            ;
        call    wrtbl           ;
cmd8:   cmp     al,'8'          ;Read FSA status
        jnz     cmd9            ;
        call    rdfsa           ;
cmd9:   cmp     al,'9'          ;Abort
        jnz     cmda            ;
        call    abortc          ;
cmda:   cmp     al,'A'          ;Write seek
        jnz     cmdb            ;
        call    wrtsk           ;
cmdb:   cmp     al,'B'          ;Read bootloop
        jnz     cmdc            ;
        call    rdbl            ;
cmdc:   cmp     al,'C'          ;Read corrected data
        jnz     cmdd            ;
        call    rlcd            ;
cmdd:   cmp     al,'D'          ;Reset fifo
        jnz     cmde            ;
        call    resetf          ;
cmde:   cmp     al,'E'          ;MBM purge
        jnz     cmdf            ;
        call    purge           ;
cmdf:   cmp     al,'F'          ;Sortware reset
        call    reset           ;
        ret
;
conin:
```

```
        mov     cl,01h  ;
        int     224     ;
        ret             ;
;****************************************
;*                                      *
;*  This routines writes 0e5h into each byte *
;*  in the bubble memory system.  This is    *
;*  preparatory to using the bubble as a     *
;*                  disk                     *
;*                                      *
;****************************************
;
formatb:
        mov     dx,2047         ;set counter equal to # of
        mov     ax,0001h        ;load block length
        mov     blklen,ax       ;block length equal 1
        mov     al,02h          ;load nfc value
        mov     nfc,al          ;nfc equal 2
        mov     al,20h          ;load enable register value
        mov     enable,al       ;enable equal 20h(level 1 E
        mov     al,00h          ;clear al
        mov     bblnum,al       ;bblnum equal 0
format1:
        mov     al,1dh          ;load reset fifo cmd
        out     cmdsts,al       ;send it
        mov     ax,dx           ;move value in dx to ax
        mov     pageno,ax       ;page address equal to valu
        call    wtreg1          ;load the parametric regist
        mov     bx,128          ;bx equal byte count
        push    dx              ;save dx value
        mov     si,offset frmbuf        ;set pointer
        call    writea          ;write a 128 byte page
        pop     dx              ;retrieve  dx value
        dec     dx              ;decrement dx by one
        cmp     dx,0            ;compare dx to zero
        jnz     format1         ;if not zero go again
        ret
;
blklen  rw      1               ;
nfc     rb      1               ;
enable  rb      1               ;
pageno  rw      1               ;
bblnum  rb      1               ;
;
table1 db       00,00,01,20h,03h,0ffh,01
;
table2  rb      7
;
table3  rb      7
;
```

```
temptbl rb      12
;
temp1   rb      1
;
temp2   rw      1
;
temp3   rw      1
;
datbuf  rb      128
;
frmbuf  db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h,0e5h
        db      0e5h,0e5h
;
wrtbuf  db      00h,01h,02h,03h,04h,05h,06h,07h,08h,09h
        db      0ah,0bh,0ch,0dh,0eh,0fh,10h,11h,12h,13h
        db      14h,15h,16h,17h,18h,19h,1ah,1bh,1ch,1dh
        db      1eh,1fh,20h,21h,22h,23h,24h,25h,26h,27h
        db      28h,29h,2ah,2bh,2ch,2dh,2eh,2fh,30h,31h
        db      32h,33h,34h,35h,36h,37h,38h,39h,3ah,3bh
        db      3ch,3dh,3eh,3fh,40h,41h,42h,43h,44h,45h
        db      46h,47h,48h,49h,4ah,4bh,4ch,4dh,4eh,4fh
        db      50h,51h,52h,53h,54h,55h,56h,57h,58h,59h
        db      5ah,5bh,5ch,5dh,5eh,5fh,60h,61h,62h,63h
        db      64h,65h,66h,67h,68h,69h,6ah,6bh,6ch,6dh
        db      6eh,6fh,70h,71h,72h,73h,74h,75h,76h,77h
        db      78h,79h,7ah,7bh,7ch,7dh,7eh,7fh
tempst  rb      1
;
status  db      cr,lf,' ','X','X','X','X','X','X','X','X',lf
;
msg1    db      cr,lf,'      Menu for Bubble Memory Controll
        db      cr,lf,'          select one function'
        db      cr,lf,lf,' 1 - Abort Command'
        db      cr,lf,' 2 - Send Any Command'
        db      cr,lf,' 3 - Get Bubble Memory Status'
        db      cr,lf,' 4 - Format Bubble Memory',cr,lf,'$'
;
```

```
msg2    db      cr,lf,'Operation Complete','$'
;
msg3    db      cr,lf,'Operation Failed','$'
;
msg4    db      cr,lf,'Time Out Failure','$'
;
msg5    db      cr,lf,'No Response','$'
;
msg6    db      cr,lf,'Abort Fail','$'
;
msg7    db      cr,lf,'Enter Parametric Register values','$
;
msg8    db      cr,lf,'Block Length 0-2047 (enter 4 digits)
;
msg9    db      cr,lf,'Number of Channels 0-4 (enter 1 digi
;
msg10   db      cr,lf,'Set Enable Register 1-99 (enter 2 di
;
msg11   db      cr,lf,'Page Number 0-2047 (enter 4 digits)'
;
msg12   db      cr,lf,'Bubble Number 0-3 (enter 1 digit)','
;
msg13   db      cr,'This is the Status Byte',cr,lf,'$'
;
msg14   db      cr,lf,'  Menu for Command Selection  '
        db      cr,lf,'          Select One          '
        db      cr,lf,lf,' 0 - Write Bootlocp Register Mask
        db      cr,lf,' 1 - Initialize'
        db      cr,lf,' 2 - Read Bubble Data'
        db      cr,lf,' 3 - Write Bubble Data'
        db      cr,lf,' 4 - Read Seek'
        db      cr,lf,' 5 - Read Bootloop Register'
        db      cr,lf,' 6 - Write Bootloop Register'
        db      cr,lf,' 7 - Write Bootloop'
        db      cr,lf,' 8 - Read FSA Status'
        db      cr,lf,' 9 - Abort'
        db      cr,lf,' A - Write Seek'
        db      cr,lf,' B - Read Bootloop'
        db      cr,lf,' C - Read Corrected Data'
        db      cr,lf,' D - Reset FIFO'
        db      cr,lf,' E - MBM Purge'
        db      cr,lf,' F - Software Reset',cr,lf,'$'
;
        end
```

APPENDIX B
PROGRAM LISTING OF SINGLES.DEF

```
disks 3
diskdef 0,1,26,6,1024,243,64,64,2
diskdef 1,0
diskdef 2,1,26,0,1024,243,64,64,2
endef
```

APPENDIX C
PROGRAM LISTING OF SINGLES.LIB

```
;       DISKS 3
dpbase  equ     $               ;Base of Disk Parameter Bloc
dpeØ    dw      xltØ,ØØØØh      ;Translate Table
        dw      ØØØØh,ØØØØh     ;Scratch Area
        dw      dirbuf,dpbØ     ;Dir Buff, Parm Block
        dw      csvØ,alvØ       ;Check, Alloc Vectors
dpe1    dw      xlt1,ØØØØh      ;Translate Table
        dw      ØØØØh,ØØØØh     ;Scratch Area
        dw      dirbuf,dpb1     ;Dir Buff, Parm Block
        dw      csv1,alv1       ;Check, Alloc Vectors
dpe2    dw      xlt2,ØØØØh      ;Translate Table
        dw      ØØØØh,ØØØØh     ;Scratch Area
        dw      dirbuf,dpb2     ;Dir Buff, Parm Block
        dw      csv2,alv2       ;Check, Alloc Vectors
;       DISKDEF Ø,1,26,6,1Ø24,243,64,64,2
dpbØ    equ     offset $        ;Disk Parameter Block
        dw      26              ;Sectors Per Track
        db      3               ;Block Shift
        db      7               ;Block Mask
        db      Ø               ;Extnt Mask
        dw      242             ;Disk Size - 1
        dw      63              ;Directory Max
        db      192             ;AllocØ
        db      Ø               ;Alloc1
        dw      16              ;Check Size
        dw      2               ;Offset
xltØ    equ     offset $        ;Translate Table
        db      1,7,13,19
        db      25,5,11,17
        db      23,3,9,15
        db      21,2,8,14
        db      2Ø,26,6,12
        db      18,24,4,1Ø
        db      16,22
alsØ    equ     31              ;Allocation Vector Size
cssØ    equ     16              ;Check Vector Size
;       DISKDEF 1,Ø
dpb1    equ     dpbØ            ;Equivalent Parameters
als1    equ     alsØ            ;Same Allocation Vector Size
css1    equ     cssØ            ;Same Checksum Vector Size
xlt1    equ     xltØ            ;Same Translate Table
;       DISKDEF 2,1,26,Ø,1Ø24,243,64,64,2
dpb2    equ     offset $        ;Disk Parameter Block
```

```
        dw      26                  ;Sectors Per Track
        db      3                   ;Block Shift
        db      7                   ;Block Mask
        db      0                   ;Extnt Mask
        dw      242                 ;Disk Size - 1
        dw      63                  ;Directory Max
        db      192                 ;Alloc0
        db      0                   ;Alloc1
        dw      16                  ;Check Size
        dw      2                   ;Offset
xlt2    equ     offset $            ;Translate Table
        db      1,2,3,4
        db      5,6,7,8
        db      9,10,11,12
        db      13,14,15,16
        db      17,18,19,20
        db      21,22,23,24
        db      25,26
als2    equ     31                  ;Allocation Vector Size
css2    equ     16                  ;Check Vector Size
;       ENDEF
;
;       Uninitialized Scratch Memory Follows:
;
begdat  equ     offset $            ;Start of Scratch Area
dirbuf  rs      128                 ;Directory Buffer
alv0    rs      als0                ;Alloc Vector
csv0    rs      css0                ;Check Vector
alv1    rs      als1                ;Alloc Vector
csv1    rs      css1                ;Check Vector
alv2    rs      als2                ;Alloc Vector
csv2    rs      css2                ;Check Vector
enddat  equ     offset $            ;End of Scratch Area
datsiz  equ     offset $-begdat     ;Size of Scratch Area
        db      0                   ;Marks End of Module
```

APPENDIX D
PROGRAM LISTING OF BUBBIOS.A86

```
        title   'Customized Basic I/O System'

;*********************************************
;*                                           *
;* This Customized BIOS adapts CP/M-86 to    *
;* the following hardware configuration      *
;*     Processor:  iSBC 8612                 *
;*     Controller:  iSBC 201                 *
;*     iSBC 254 Bubble Memory                *
;*     Memory model:  8080                   *
;*     Programmer: Gary Theis                *
;*                                           *
;*     Revisions :                           *
;*                                           *
;*********************************************

true            equ -1
false           equ not true
cr              equ 0dh ;carriage return
lf              equ 0ah ;line feed
max_retries     equ 10  ;for disk i/o, before perm error

;*********************************************
;*                                           *
;* Loader_bios is true if assembling the     *
;* LOADER_BIOS, otherwise BIOS is for the    *
;* CPM.SYS file.                             *
;*                                           *
;*********************************************

LOADER_BIOS     EQU FALSE
bdos_int        equ 224 ;reserved BDOS interrupt

        IF      not loader_bios
;--------------------------------------------
;|                                          |
bios_code       equ 2500h
ccp_offset      equ 0000h
bdos_ofst       equ 0B06h ;BDOS entry point
;|                                          |
;--------------------------------------------
        ENDIF   ;not loader_bios
```

```
        IF      loader_bios
;----------------------------------------------
;|                                            |
bios_code       equ 1200h ;start of LDBIOS
ccp_offset      equ 0003h ;base of CPMLOADER
bdos_ofst       equ 0406h ;stripped BDOS entry
;|                                            |
;----------------------------------------------
        ENDIF   ;loader_bios

csts    equ 0dah        ;I8251 status port
cdata   equ 0d8h        ;   "     data   "
;
;
;**********************************************
;*                                            *
;* INTEL iSBC 201 Disk Controller Ports       *
;*                                            *
;**********************************************


base    equ     078h
rtype   equ     base+1
rbyte   equ     base+3
reset   equ     base+7

dstat   equ     base
ilow    equ     base+1
ihigh   equ     base+2

;**********************************************
;*                                            *
;* INTEL iSBC 254 Bubble Memory Ports         *
;*             and Equates                    *
;*                                            *
;**********************************************

        blrpt   equ     0bh             ;pointer to first
        cmdsts  equ     0fh             ;command/status
        datreg  equ     0eh             ;data port
        nfc     equ     02h             ;number of chann
        enable  equ     20h             ;enable register
        bblnum  equ     00h             ;sets bubble sel
        blklen  equ     01h             ;sets 128 byte bl
        abtcmd  equ     19h             ;abort command
        intcmd  equ     11h             ;initialize comm
        rdcmd   equ     12h             ;read command
        wrtcmd  equ     13h             ;write command
        frcmd   equ     1dh             ;fifo reset comm
;
```

```
        IF      not loader_bios
;---------------------------------------------
;|                                           |
        ; This is a BIOS for the CPM.SYS file.
        ; Setup all interrupt vectors in low
        ; memory to address trap

        call    abort     ;abort the bubble
        call    initb     ;initialize the bubble
        push ds           ;save the DS register
        mov IOBYTE,0      ;clear IOBYTE
        mov ax,0
        mov ds,ax
        mov es,ax         ;set ES and DS to zero
        ;setup interrupt 0 to address trap routine
        mov int0_offset,offset int_trap
        mov int0_segment,CS
        mov di,4
        mov si,0          ;then propagate
        mov cx,510        ;trap vector to
        rep movs ax,ax    ;all 256 interrupts
        ;BDOS offset to proper interrupt
        mov bdos_offset,bdos_ofst
        pop ds            ;restore the DS register

;       (additional CP/M-86 initialization)
;|                                           |
;---------------------------------------------
        ENDIF   ;not loader_bios

        IF      loader_bios
;---------------------------------------------
;|                                           |
        ;This is a BIOS for the LOADER
        push ds           ;save data segment
        mov ax,0
        mov ds,ax         ;point to segment zero
        ;BDOS interrupt offset
        mov bdos_offset,bdos_ofst
        mov bdos_segment,CS ;bdos interrupt segment
;       (additional LOADER initialization)
        pop ds            ;restore data segment
;|                                           |
;---------------------------------------------
        ENDIF   ;loader_bios

        mov bx,offset signon
        call pmsg         ;print signon message
        mov cl,0          ;default to dr A: on coldstart
        jmp ccp           ;jump to cold start entry of CCP
```

```
        cseg
        org     ccpoffset
ccp:
        org     bios_code

;*********************************************
;*                                           *
;* BIOS Jump Vector for Individual Routines  *
;*                                           *
;*********************************************

 jmp INIT       ;Enter from BOOT ROM or LOADER
 jmp WBOOT      ;Arrive here from BDOS call 0
 jmp CONST      ;return console keyboard status
 jmp CONIN      ;return console keyboard char
 jmp CONOUT     ;write char to console device
 jmp LISTOUT    ;write character to list device
 jmp PUNCH      ;write character to punch device
 jmp READER     ;return char from reader device
 jmp HOME       ;move to trk 00 on cur sel drive
 jmp SELDSK     ;select disk for next rd/write
 jmp SETTRK     ;set track for next rd/write
 jmp SETSEC     ;set sector for next rd/write
 jmp SETDMA     ;set offset for user buff (DMA)
 jmp READ       ;read a 128 byte sector
 jmp WRITE      ;write a 128 byte sector
 jmp LISTST     ;return list status
 jmp SECTRAN    ;xlate logical->physical sector
 jmp SETDMAB    ;set seg base for buff (DMA)
 jmp GETSEGT    ;return offset of Mem Desc Table
 jmp GETIOBF    ;return I/O map byte (IOBYTE)
 jmp SETIOBF    ;set I/O map byte (IOBYTE)

;*********************************************
;*                                           *
;* INIT Entry Point, Differs for LDBIOS and  *
;* BIOS, according to "Loader_Bios" value    *
;*                                           *
;*********************************************

INIT:   ;print signon message and initialize hardware
        mov ax,cs           ;we entered with a JMPF so use
        mov ss,ax           ;CS: as the initial value of SS:
        mov ds,ax           ;DS:,
        mov es,ax           ;and ES:
        ;use local stack during initialization
        mov sp,offset stkbase
        cld                 ;set forward direction
```

```
WBOOT:  jmp ccp+6         ;direct entry to CCP at command 1

        IF      not loader_bios
;------------------------------------------------------
;|                                                    |
int_trap:
        cli               ;block interrupts
        mov ax,cs
        mov ds,ax         ;get our data segment
        mov bx,offset int_trp
        call pmsg
        hlt               ;hardstop
;|                                                    |
;------------------------------------------------------
        ENDIF   ;not loader_bios

;*****************************************************
;*                                                   *
;*   CP/M Character I/O Interface Routines           *
;*                                                   *
;*    console is USART (i8251A) on iSBC 8612         *
;*               at ports D8/DA                      *
;*****************************************************

CONST:          ;console status
        in al,csts
        and al,2
        jz const_ret
        or al,255         ;return non-zero if rda
const_ret:
        ret               ;rcvr data available

CONIN:                    ;console input
        call CONST
        jz CONIN          ;wait for RDA
        in al,cdata
        and al,7fh        ;read data & remove parity bit
        ret

CONOUT:         ;console output
        in al,csts
        and al,1          ;get console status
        jz CONOUT
        mov al,cl
        out cdata,al      ;transmitter buffer is empty
        ret               ;then return data

LISTOUT:                  ;list device output
                          ;not implemented
        ret
```

```
LISTST:                         ;poll list status
                                ;not implemented
        ret

PUNCH:                  ;write punch device
                                ;not implemented

READER:
        mov al,1ah              ;return eof for now
        ret

GETIOBF:
        MOV AL,IOBYTE           ;IOBYTE NOT IMPLEMENTED
        ret

SETIOBF:
        MOV IOBYTE,CL
        ret                     ;iobyte not implemented

; Routine to get and echo a console character
;       and shift it to upper case

uconecho:
        call CONIN              ;get a console character
        push ax
        mov cl,al               ;save and
        call CONOUT
        pop ax                  ;echo to console
        cmp al,'a'
        jb uret                 ;less than 'a' is ok
        cmp al,'z'
        ja uret                 ;greater than 'z' is ok
        sub al,'a'-'A'          ;else shift to caps
uret:
        ret
pmsg:
        mov al,[BX]             ;get next char from message
        test al,al
        jz return               ;if zero return
        mov CL,AL
        call CONOUT             ;print it
        inc BX
        jmps pmsg               ;next character and loop

;*********************************************
;*                                           *
;*        Disk Input/Output Routines         *
;*                                           *
;*********************************************
```

```
SELDSK:         ;select disk given by register CL
ndisks  equu     3 ;number of disks (up to 16)
        mov disk,cl       ;save disk number
        mov bx,0000h      ;ready for error return
        cmp cl,ndisks     ;n beyond max disks?
        jnb return        ;return if so
        mov ch,0          ;double(n)
        mov bx,cx         ;bx = n
        mov cl,4          ;ready for *16
        shl bx,cl         ;n = n * 16
        mov cx,offset dpbase
        add bx,cx         ;dpbase + n * 16
return: ret               ;bx = .dph

HOME:   ;move selected disk to home position (Track 0)
        mov     trk,0               ;zero the track number
        mov     al,disk             ;get disk number
        cmp     al,2                ;check if its the bubble
        jz      ret_hme             ;skip if so
        mov io_com,homcom
        call execute
ret_hme:        ret

SETTRK: ;set track address given by CL
        mov trk,CL
        ret

SETSEC: ;set sector number given by cl
        mov sect,CL
        ret

SECTRAN: ;translate sector CX using table at [DX]
        mov ch,0
        mov bx,cx
        add bx,dx         ;add sector to tran table address
        mov bl,[bx]       ;get logical sector
        ret

SETDMA: ;set DMA offset given by CX
        mov dma_adr,CX
        ret

SETDMAB: ;set DMA segment given by CX
        mov dma_seg,CX
        ret
;
GETSEGT:  ;return address of physical memory table
        mov bx,offset seg_table
        ret
```

```
;*********************************************
;*                                           *
;*  All disk I/O parameters are setup:       *
;*     DISK      is disk number      (SELDSK) *
;*     TRK       is track number     (SETTRK) *
;*     SECT      is sector number    (SETSEC) *
;*    DMA_ADR  is the DMA lsb offset         *
;*  READ reads the selected sector to the DMA*
;*  address, and WRITE writes the data from  *
;*  the DMA address to the selected sector   *
;*                                           *
;*********************************************

READ:
        mov     al,disk           ;get disk number
        cmp     al,2              ;is it the bubble
        jz      skprd             ;if so skip to bubble read
        mov cl,4
        sal al,cl       ;combine disk select with opcode
        or al,rdcode
        mov io_com,al   ;create iopb
        jmps execute
skprd:  jmp bubrd       ;jump to bubble read

WRITE:
        mov     al,disk           ;get disk number
        cmp     al,2              ;is it the bubble
        jz      skpwrt            ;if so jump to bubble write
        mov cl,4
        sal al,cl
        or al,wrcode    ;create iopb for write
        mov io_com,al
        jmps    execute           ;execute disk routine
skpwrt: jmp     bubwrt            ;jump to bubble write
EXECUTE:

outer_retry:
        mov rtry_cnt,max_retries

retry:
        in al,rtype     ;clear controller
        in al,rbyte     ;
        call sendcom

idle:   in al,dstat     ;wait for completion
        and al,4        ;ready
        jz idle

;       check i.o. completion ok
        in al,rtype
```

```
;       00 unlinked i/o complete        01 linked i/o comp.
;       10 disk status changed          11 (not used)
;       must be a 00 in al
        test al,10b       ;ready status change?
        JNZ WREADY
        OR AL,0
        jnz werror        ;some other error, retry

;       check i/o error bits
        in al,rbyte
        rcl al,1
        mov err_code,80h
        jb wready         ;unit not ready
        rcr al,1
        mov err_code,al
        and al,0feh       ;any other errors?
        jnz werror
;
;       read or write is ok, al contains 0
        ret

wready: ;not ready, treat as an error for now
        in al,rbyte       ;clear result byte
        jmps trycount

werror: ;return hardware malfunction
trycount:
        dec rtry_cnt
        jnz retry
        mov al,err_code
        mov ah,0
        mov bx,ax         ;make error code 16 bits
        mov bx,errtbl[BX]
        call pmsg         ;print appropriate message
        in al,cdata       ;flush usart receiver buffer
        call uconecho     ;read upper case console charact
        cmp al,'C'
        je wboot_1        ;cancel
        cmp al,'R'
        je outer_retry    ;retry 10 more times
        cmp al,'I'
        je z_ret          ;ignore error
        or al,255         ;set code for permanent error
z_ret:  ret


wboot_1:                  ;can't make it w/ a short leap
        jmp WBOOT
```

```
;*********************************************************
;*                                                       *
;*  sendcom sends the address of the iopb to             *
;*    the iSBC 201                                       *
;*                                                       *
;*********************************************************

sendcom:
        MOV CL,4
        MOV AX,DMA_SEG
        SAL AX,CL
        ADD AX,DMA_ADR
        MOV IO_ADR,AX
        MOV CL,4
        MOV AX,CS
        SAL AX,CL
        ADD AX,OFFSET CHANCMD ;ADD SEG & OFFSET FOR 201
        out ilow,al
        mov cl,8
        sar ax,cl
        out ihigh,al
        ret
;
;*****************************************************
;*                                                   *
;* This routine reads a 128 byte sector from         *
;*          the bubble memory module.                *
;*                                                   *
;*****************************************************
bubrd:
        mov     ah,0              ;clear ah register
        mov     al,trk            ;get track number
        mul     constn            ;multiply by 26
        xor     dx,dx             ;clear dx
        add     dl,sect           ;add sector number
        add     ax,dx             ;add total
        mov     pageno,ax         ;this is the page number
        call    wtreg             ;write parametric regist
        push    es                ;save extra segment
        mov     ax,dma_seg        ;set the extra segment
        mov     es,ax             ;equal to the dma_seg.
        mov     bx,128            ;128 bytes to be read
        mov     di, dma_adr       ;set pointer
        mov     al,rdcmd          ;get read command
        out     cmdsts,al         ;send it
        mov     cx,0ffffh         ;set timeout counter
bread1:
        in      al,cmdsts         ;get status
        test    al,80h            ;test for busy
        loopz   bread1            ;wait for busy
```

```
        jcxz    error1              ;timeout error
        mov     cx,bx               ;load # of bytes to read
bread2:
        in      al,cmdsts           ;get status
        test    al,01h              ;test for fifo empty
        jz      bread3              ;yes,check for busy
        in      al,datreg           ;no,get data
        stos    al                  ;store it
        loop    bread2              ;go again
        jmp     pause               ;wait for good status
bread3:
        mov     dx,0ffffh           ;timeout in dx reg
        test    al,80h              ;check for busy
        jz      skip1               ;
        dec     dx                  ;decrement timeout cntr
        cmp     dx,0                ;compare timeout to 0
        jnz     bread2              ;still busy wait
skip1:  sub     bx,cx               ;bytes trans in bx
        mov     temp3,bx            ;store byte transfer cnt.
        jmp     error2
pause:
        in      al,cmdsts           ;get status
        test    al,80h              ;check for busy
        jz      contin              ;not busy-send status
        mov     cx,0ffffh           ;set up timeout cntr.
pollb:
        in      al,cmdsts           ;get status
        test    al,80h              ;check for busy
        loopnz  pollb               ;wait for busy to clear
        jcxz    error1              ;but not too long
contin:
        xor     al,al               ;return a 0 in al
        pop     es                  ;return extra segment
        ret
error1:
        mov     bx,offset buberr1       ;timeout error
        call    pmsg                ;print the message
        call    abort               ;abort the bubble
        call    initb               ;initialize the bubble
        mov     al,01h              ;return with a 1 in al
        pop     es                  ;return extra segment
                ret
error2:
        mov     bx,offset buberr2       ;read op fail
        call    pmsg                ;print the message
        call    abort               ;abort the bubble
        call    initb               ;initialize the bubble
        mov     al,01h              ;return with a 1 in al
        pop     es                  ;return extra segment
        ret
```

```
;*********************************************
;*                                           *
;*  This routine writes a 128 byte sector    *
;*        into the bubble memory             *
;*                                           *
;*********************************************
bubwrt:
        mov     ah,0            ;clear ah register
        mov     al,trk          ;get track number
        mul     constn          ;multiply by 26
        xor     dx,dx           ;clear dx
        add     dl,sect         ;add sector number
        add     ax,dx           ;combine the total
        mov     pageno,ax       ;this is the page number
        call    wtreg           ;write parametric regist
        mov     ax,dma_seg      ;set the data segment
        mov     si,dma_adr      ;set pointer
        push    ds              ;save data segment
        mov     ds,ax           ;equal to the dma_seg
        mov     bx,128          ;number of bytes to be
        mov     al,wrtcmd       ;load write command
        out     cmdsts,al       ;send it
        mov     cx,0ffffh       ;timeout cntr
bwrite1:
        in      al,cmdsts       ;get status
        test    al,80h          ;check for busy
        loopz   bwrite1         ;wait for busy
        jcxz    error11         ;timeout error
        test    al,01h          ;test for fifo ready
        loopz   bwrite1         ;wait for fifo
        jcxz    error11         ;timeout error
        mov     cx,bx           ;load # number of bytes
bwrite2:
        in      al,cmdsts       ;get status
        test    al,01h          ;check for fifo ready
        jz      bwrite3         ;no-wait
        lods    al              ;yes-get data
        out     datreg,al       ;send data to bubble
        loop    bwrite2         ;go again
        jmp     pause1          ;return good status
bwrite3:
        mov     dx,0ffffh       ;set timeout cntr
        test    al,80h          ;check ofr busy
        jz      skip11          ;
        dec     dx              ;decrement timeout cntr.
        cmp     dx,0            ;compare timeout to zero
        jnz     bwrite2         ;try again
skip11: sub     bx,cx           ;# of bytes transferred
        jmp     error22         ;op fail error
pause1:
        in      al,cmdsts       ;get status
```

```
        test    al,80h              ;check for busy
        jz      contin1             ;return op complete
        mov     cx,0ffffh           ;set timeout
pollb1:
        in      al,cmdsts           ;get status
        test    al,80h              ;check for busy
        loopnz  pollb1              ;but not too long
        jcxz    error11             ;but not too long
contin1:
        xor     al,al               ;return a 0 in al
        pop     ds                  ;return the data segment
        ret
error11:
        mov     bx,offset buberr3       ;timeout failure
        call    pmsg
        call    abort               ;abort the bubble
        pop     ds                  ;return the data segment
        call    initb               ;initialize the bubble
        mov     al,01h              ;return with a 1 in al
        ret
error22:
        mov     bx,offset buberr4       ;op fail msg
        call    pmsg
        call    abort               ;abort the bubble
        pop     ds                  ;return the data segment
        call    initb               ;initialize the bubble
        mov     al,01h              ;return with a 1 in al
        ret
;*********************************************
;*                                           *
;*  This routine aborts the bubble memory.   *
;*                                           *
;*********************************************
abort:
        mov     cx,0ffffh           ;init timeout cntr
        mov     bh,40h              ;load op comp bit
        mov     al,abtcmd           ;load abort command
        out     cmdsts,al           ;issue it
busy:
        in      al,cmdsts           ;get status
        and     al,80h              ;check for busy
        jz      poll                ;if busy jump to poll
        dec     cx                  ;else decrement cx
        xor     ax,ax               ;clear ax reggg
        cmp     cx,ax               ;heck timeout count = 0
        jnz     busy                ;time left,try busy again
        jmp     ret1                ;return with error
poll:
        in      al,cmdsts           ;get status
        test    al,40h              ;check for status=40h
        jnz     ret2                ;return with op comp
```

```
        dec     cx                      ;else decrement timeout cntr
        xor     ax,ax                   ;clear ax
        cmp     cx,ax                   ;compare timeout to 0
        jnz     poll                    ;try again
retl:
        in      al,cmdsts               ;get status
        mov     bx, offset buberr5          ;abort fail msg
        call    pmsg
        mov     al,1                    ;return a 1 in al
        pop     bx
        ret                             ;return to system
ret2:
        mov     dx,27600                ;delay 100ms
loop:
        xor     ax,ax                   ;clear ax
        dec     dx                      ;decrement count
        cmp     dx,ax                   ;is it zero
        jnz     loop                    ;no-go again
        ret                             ;yes-return
;
;*********************************************
;*                                           *
;* This routine writes the values into the   *
;* bubble memory parametric registers.       *
;*                                           *
;*********************************************
;
wtreg:
        mov     al,blrpt                ;set register pointer
        out     cmdsts,al               ;set pointer in bmc
        mov     bx,blklen               ;load block length
;
        mov     al,bl                   ;this series of instructions
        out     datreg,al               ;combines block length
        mov     al,nfc                  ;and the nfc value
        mov     cl,4                    ;to form a sixteen bit
        shl     al,cl                   ;word to place in
        or      al,bh                   ;the block length
        out     datreg,al               ;register
;
        mov     al,enable               ;load enable register values
        out     datreg,al               ;and send to bmc
;
        mov     bx,pageno               ;load pageno
;
        mov     al,bl                   ;this series of instructions
        out     datreg,al               ;combines page number
        mov     al,bblnum               ;and bubble number
        mov     cl,3                    ;to form a sixteen bit
        shl     al.cl                   ;word to place in
```

```
        or      al,bh               ;the address
        out     datreg,al           ;register
        ret
;
;*******************************************
;*                                         *
;* This routine initializes the bubble memory *
;*                                         *
;*******************************************
;
initb:
        mov     al,blrpt            ;set pointer
        out     cmdsts,al           ;set pointer in bmc
        mov     bx,0000h            ;load init block length
;
        mov     al,bl               ;this series of instructions
        out     datreg,al           ;combines block length
        mov     al,01               ;and the nfc value
        mov     cl,4                ;to form a sixteen bit
        shl     al,cl               ;word to place in
        or      al,bh               ;the block length
        out     datreg,al           ;register
;
        mov     al,20h              ;load enable register values
        out     datreg,al           ;and send to bmc
;
        mov     bx,pageno           ;load init page number
;
        mov     al,bl               ;this series of instructions
        out     datreg,al           ;combines page number
        mov     al,01h              ;and bubble number
        mov     cl,3                ;to form a sixteen bit
        shl     al,cl               ;word to place in
        or      al,bh               ;the address
        out     datreg,al           ;register
;
        mov      cx,0ffffh          ;set timeout cntr
        mov     bx,40h              ;load op comp status bit
        mov     al,intcmd           ;load initialize cmd
        out     cmdsts,al           ;send it
busy1:
        in      al,cmdsts           ;get status
        and     al,80h              ;check for busy
        jz      poll1               ;if not busy jump
        dec     cx                  ;decrement timeout cntr
        xor     ax,ax               ;clear ax
        cmp     cx,ax               ;compare timeout to zer0
        jnz     busy1               ;try again if time left
        jmp     retal               ;timeout failure
poll1:
```

```
        in      al,cmdsts           ;get status
        xor     al,40h              ;check if opcomp
        jz      reta2               ;if complete return
        dec     cx                  ;decrement timeout cntr
        xor     ax,ax               ;clear ax
        cmp     cx,ax               ;compare timeout to 0
        jnz     poll1               ;try again
retal:
        mov     bx,offset buberr6       ;timeout failure
        call    pmsg                ;
        pop bx                      ;
        ret                         ;return to system
reta2:
        ret                         ;return to routine
;
;*********************************************
;*               Data Areas                  *
;*                                           *
;*********************************************
data_offset     equ offset $

        dseg
        org     data_offset         ;contiguous with code seg
IOBYTE  db      0
disk    db      0       ;disk number
chancmd db      80h     ;iopb channel word
io_com  db      0
nsec    db      1       ;number sectors to xfer
trk     db      0
sect    db      0       ;start sector
IO_ADR  DW      0000H   ;PHYS ADDR FOR SBC201 USE
dma_adr dw      0080h   ;DMA adr (default)
dma_seg dw      0       ;DMA Base Segment
pageno  dw      0       ;page number for bubble memory
temp3   rw      1       ;temporary storage
constn dw       001ah   ;26 sectors per track

BOM_COM EQU 3
RDCODE  EQU 4
ERR_CODE DB 00H
WRCODE  EQU 6

        IF      loader_bios
;-------------------------------------------
;|                                         |
signon  db      cr,lf,cr,lf
        db      'CP/M-86 Version 1.0',cr,lf,0
;|                                         |
;-------------------------------------------
        ENDIF   ;loader_bios
```

```
        IF      not loader_bios
;-----------------------------------------------
;|                                             |
signon  db      cr,lf,cr,lf
        db      'System Generated 2/22/84'
        db      cr,lf,0
;|                                             |
;-----------------------------------------------
        ENDIF   ;not loader_bios

int_trp db      cr,lf
        db      'Interrupt Trap Halt'
        db      cr,lf,0

errtbl  dw er0,er1,er2,er3
        dw er4,er5,er6,er7
        dw er8,er9,erA,erB
        dw erC,erD,erE,erF
        dw er10,er20,er40,er80

er0     db  cr,lf,'Null Error ??',0
er1     db  cr,lf,'Deleted Record :',0
er2     db  cr,lf,'CRC Error :',0
er3     db  cr,lf,'Data Overrun-Underrun :',0
er4     db  cr,lf,'Seek Error :',0
er5     equ er0
er6     equ er0
er7     equ er0
er8     db  cr,lf,'Address Error :',0
er9     db  cr,lf,'Write Protect :',0
erA     db  cr,lf,'ID CRC Error :',0
erB     db  cr,lf,'Write Error :',0
erC     db  cr,lf,'Sector Not Found :',0
erD     equ er0
erE     db  cr,lf,'No Address Mark :',0
erF     db  cr,lf,'Data Mark Error :',0
er10    equ er3
er20    equ er9
er40    equ erB
er80    db  cr,lf,'Drive Not Ready :',0
buberr1 db      cr,lf,'Bubble Read Timeout Error:',0
buberr2 db      cr,lf,'Bubble Read Failure:',0
buberr3 db      cr,lf,'Bubble Write Timeout Error:',0
buberr4 db      cr,lf,'Bubble Write Failure:',0
buberr5 db      cr,lf,'Bubble Abort Failure:',0
buberr6 db      cr,lf,'Bubble Initialize Failure:',0

rtry_cnt db 0   ;disk error retry counter
```

```
;       System Memory Segment Table

segtable db 1   ;1 segments
        dw tpa_seg      ;1st seg starts after BIOS
        dw tpa_len      ;and extends to 08000

        include singles.lib ;read in disk definitions

loc_stk rw  32  ;local stack for initialization
stkbase equ offset $

lastoff equ offset $
tpa_seg equ (lastoff+0400h+15) / 16
tpa_len equu 0F00h - tpa_seg
        db 0    ;fill last address for GENCMD

;*********************************************
;*                                           *
;*          Dummy Data Section               *
;*                                           *
;*********************************************
        dseg    0       ;absolute low memory
        org     0       ;(interrupt vectors)
int0_offset     rw      1
int0_segment    rw      1
;       pad to system call vector
        rw      2*(bdos_int-1)

bdos_offset     rw      1
bdos_segment    rw      1
        END


rtry_cnt db 0   ;disk error retry counter
```

LIST OF REFERENCES

1. Giess, Edward A., "Magnetic Bubble Materials," Science, V. 208, pp. 938-943, May 23, 1980.

2. Luhn, Robert, "A Lasting Memory," PC World, V. 1, no. 6, pp. 50-58, SEP 1983.

3. Chang, H., Magnetic Bubble Technology, IEEE Press, 1975.

4. Intel Corporation, Memory Components Handbook, 1983.

5. Mims, Forrest M., "Solid State Developments; Magnets, Bubbles, Garnets," Popular Electronics, pp. 88-90, March 1981.

6. Hicklin, M. S., Neufeld, J. A., Adaptation of Magnetic Bubble Memory in a Standard Microcomputer Environment, Masters Thesis, Naval Postgraduate School, Monterey, California, 1981.

7. "No Boom For Bubble Memories," Business Week, pp. 152-155, May 4, 1981.

8. Intel Corporation, iSBC 254 Technical Manual, 1981.

9. Intel Corporation, BPK 72 Bubble Memory Prototype Kit Users Manual, 1983.

10. Intel Corporation, INTELLEC Microcomputer Development System Reference Manual, 1976.

11. Digital Research, CP/M-86 Operating System System Guide, 1981.

12. Digital Research, CP/M-86 Operating System Programmers Guide, 1981.

13. Candalon, M. B., Alteration of the CP/M-86 Operating System, Masters Thesis, Naval Postgraduate School, Monterey, California, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
4.	Professor M. L. Cotton, Code 62Cc Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	3
5.	Professor R. Panholzer, Code 62Pz Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943	1
6.	LCDR Gary A. Theis 338 S. College Street Grenada, Mississippi 38901	1

208576

Thesis
T3635 Theis
c.1
Utilization of a Bubble Memory System as a microcomputer disk resource.